



ENVI Services Engine 5.3

User Guide

© 2015 Exelis Visual Information Solutions, Inc., a subsidiary of Harris Corporation

Table of Contents

Copyright Notices	5
Limitation of Warranty	5
Permission to Reproduce Manuals	5
Export Control Information	5
Copyright and Trademark Notices	5
What's New in This Release	6
Overview	7
Architecture	8
More About REST	9
Hardware and Network Components	10
Single Computer	10
Cluster of Computers	11
Scaling and Load Balancing	14
Installation	16
Before You Begin	17
Windows Considerations	17
Linux Considerations	18
Windows Installation Instructions	19
Start ESE as an Application	20
Linux Installation Instructions	21
Install ESE as an Application	21
Install ESE as a Daemon	22
Starting the ESE Daemon on Other Nodes	23
Uninstalling ESE from Linux RedHat Systems	23
Uninstalling ESE from Debian Linux Systems	24
Installation Components	25
IDL and ENVI Directories	25
ENVI Services Engine Executable	25
ESE_WORK_DIR	25
Licensing Overview	28

Job Processing and Node Locking	28
Configuration Options	30
Configuration File	31
Keys	31
Version History	34
IDL Reset Options	35
Global Reset	35
Task Reset in the *.task file	35
Valid Values	35
IDL Thread Pool	38
System Administration	39
Stop and Start the System	40
Manually Stop the Service	40
Manually Start the Service	40
Manage the Catalog	42
Manage Tasks	45
Upload Tasks	45
Submit Jobs	46
Monitor Jobs	48
Job Statistics	49
Job List	49
Check Job Status	50
Check Log Files	51
Master Logs	51
Node Logs	51
Job Logs	52
Monitor Usage Statistics	53
Security	55
Security Best Practices	55
Using a Web Server with ESE	56
Writing Tasks	57
Before You Begin	57
Write the PRO Code	57
Create a Task Definition File	59
Publish the Task	60
Task Definition File	61
Task Object Keys	61
Parameter Object Keys	62
System Variables	65

!SERVER	65
Instantiate a Server	68
ENVI Routines	70
Data Types	71
Version History	71
IDL and ESE Data Types	72
ENVI Data Types	74
Spatial Reference Data Types	81
Esri® Data Types	88
Writing Client Applications	89
Example IDL Client	90
Options for Selecting Input Data	93
Knowledge Base	95
Installation	95
System Administration	95
Tasks and Clients	100
Contact Us	102
Corporate Office	102
Technical Support	102
Training	102
Sales and Custom Development	102

Copyright Notices

The IDL® and ENVI® software programs and the accompanying procedures, functions, and documentation described herein are sold under license agreements. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement. Exelis Visual Information Solutions, Inc., a subsidiary of Harris Corporation (Exelis VIS), reserves the right to make changes to this document at any time and without notice.

Limitation of Warranty

Exelis VIS makes no warranties, either express or implied, as to any matter not expressly set forth in the license agreement, including without limitation the condition of the software, merchantability, or fitness for any particular purpose. Exelis VIS shall not be liable for any direct, consequential, or any other damages, suffered by the licensed user or any other users resulting from the use of the software packages or the software documentation.

Permission to Reproduce Manuals

If you are a licensed user of Exelis VIS software, Exelis VIS grants you a limited, nontransferable license to reproduce its software's manuals provided such copies are for your use only and are not sold or distributed to third parties. All such copies must contain the title page and Exelis Visual Information copyright notice.

Export Control Information

Exelis VIS Software and its associated technology are subject to U.S. export controls including the United States Export Administration Regulations. The licensed user is responsible for ensuring compliance with all applicable U.S. export control laws and regulations. These laws include restrictions on destinations, end users and end use.

Copyright and Trademark Notices

IDL® and ENVI® are trademarks of Exelis Inc., a subsidiary of Harris Corporation.

Esri®, ArcGIS®, ArcView®, and ArcInfo® are registered trademarks of Esri.

Adobe Illustrator® and Adobe PDF® Print Engine are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Macintosh® is a registered trademark of Apple Inc., registered in the U.S. and other countries.

PowerPoint®, PowerPoint icon and Windows® are registered trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX® is a registered trademark of The Open Group.

FLAASH® and QUAC® are registered trademarks for Spectral Sciences, Inc.

Other trademarks and registered trademarks are the property of the respective trademark holders.

© 2015 Exelis Visual Information Solutions, Inc., a subsidiary of Harris Corporation.

What's New in This Release

- The initialization sequence was updated to automatically fail jobs that were left over from the prior session in a non-terminal state such as "queued" and "running." The progress message field of automatically failed jobs updates with text describing why the job failed. Client applications should always monitor the status of their jobs and respond appropriately to success and failure.
- Fixed leaks and improved efficiency.

Overview

The ENVI Services Engine (ESE) is an application that provides on-demand IDL, ENVI, and ENVI LiDAR processing as a web service. Client applications make HTTP calls to invoke IDL and ENVI processing, and the results are returned to the client via standard HTTP mechanisms. Client applications can be web pages, IDL programs, or any other type of program that can make HTTP calls. ESE does not provide client functionality, leaving it free to work with any type of client. However, the ESE installation does include example clients, as well as example tasks.

The following are key components of ESE:

- It is a processing engine that is intended for:
 - Distributed processing of thousands of independent requests, or splitting up one request into discrete jobs.
 - Providing batch processing capability.
- Runs on 64-bit Linux and Windows as either a daemon or a standard application (launched manually).
- Provides the application programming interface (API) to build the back-end for clients to perform data processing.
- Contains a single, system-wide configuration file that is easy to maintain.
- Has one executable that can run in three modes: master, node leader, or worker.
- Uses a built-in database to track job execution and performance.
- Works by employing tasks that contain processing instructions.
- Acts as the processing engine for ENVI for ArcGIS - Services Edition (EFASE).

Application developers: See the following topics:

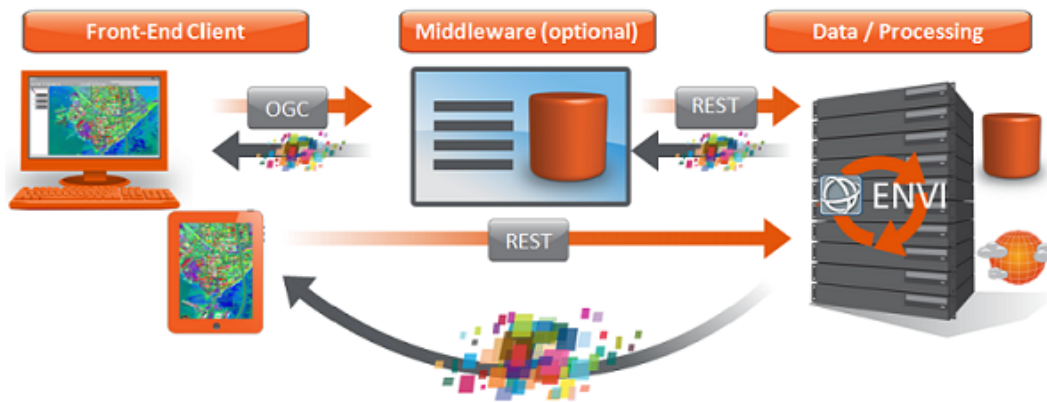
- Writing Tasks
- Writing Client Applications

System administrators: See the following topics:

- Architecture
- Hardware and Network Components
- System Administration
- Configuration Options

Architecture

The following diagram provides an overview of the ESE architecture at an enterprise level:



The *front-end client* can be a web-based thin client, desktop program, or mobile application. It is the interface used to search and discover data, request analytics, and display processing results. The front-end client makes a processing request in the form of an HTTP REpresentational State Transfer (REST) call to either the middleware or directly to the server.

The optional *middleware* component controls all of the geographic information systems (GIS) cataloging and dissemination of source and derived products. This component often employs Open Geospatial Consortium (OGC)-based architecture to handle map, feature, and process requests that are passed to it from the front-end clients. Middleware contains the ability to host and serve enormous catalogs of data as web services.

The *data and processing server* manages the data and analytic functions. Once the ESE server receives a REST call, it initiates an IDL and/or ENVI instance and runs one or more analyses on the data specified in the call. Data can be hosted locally, remotely, or even be consumed as a web service. The processing options include standard ENVI image analysis functionality, batch IDL functions, or customized workflows that are available as callable functions via front-end clients.

Once ESE finishes processing, the derived products or other requested information are delivered back to the front-end client either directly or via the middleware.

The key to the ESE architecture is that it works by moving the analytics to the data as opposed to having data transferred to a desktop for local analysis. This reduces the time it takes to execute certain types of tasks. The server component is *scalable*, which means you can employ several servers to manage large tasks or batch jobs. Also, the information returned to the client is often just a fraction of the size of the original data, making retrieving information on small devices easy. The ability to precisely extract the information needed allows the system to minimize data flow across the network.

More About REST

REpresentational State Transfer (REST) describes an architecture for distributed systems. The ESRI GeoProcessing REST specification is one implementation where communication is achieved through HTTP. The main concept of REST is that an application's state is kept on the client, not on the server. Any state required to perform a task is first transferred from the client to the server. If the server updates the state then it must pass the state back to the client. The practical effect is to achieve scalability and robustness. Since the client stores the state, it does not matter which physical server does the processing, so it is scalable. This is a key feature of ESE.

With a REST implementation, ESE will effectively launch and destroy IDL sessions with each invocation of a task. IDL and ENVI variables do not survive between tasks and for efficiency, IDL and ENVI are not restarted between jobs. However, ESE does offer the option to perform a reset after each job. ESE also offers an option not to run ENVI.

Any state that needs to be used by subsequent tasks must be passed back to the client, either as data in the HTTP response or as a URL to an addressable resource on the server. Subsequent tasks can retrieve prior state directly from data passed in the client request or from an addressable server resource specified in the client request.

The protocol used for client and server communication is HTTP. ESE also uses JavaScript Object Notation (JSON) to package data and describe tasks. Much like XML, JSON is a data interchange format for representing data in a human-readable, textual form. It is easier to write and read than XML and is well-suited for use by JavaScript, though it is language-independent.

Hardware and Network Components

ESE can be installed on a single computer or cluster of computers. Each has a different configuration of a master process, node Leaders, and workers.

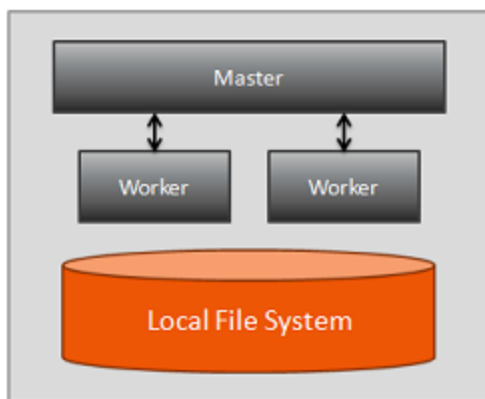
The *master process* contains an embedded web server. The master's job is to manage requests and jobs for worker processes.

The *node leader* manages worker processing but does no actual processing itself.

Workers contain an IDL interpreter and do the actual data processing. When workers complete a job, they notify the master of the results and the master returns those results to the client.

Single Computer

In this configuration, ESE is installed locally on a single computer. This structure is most often used for task/client (application) development and testing.

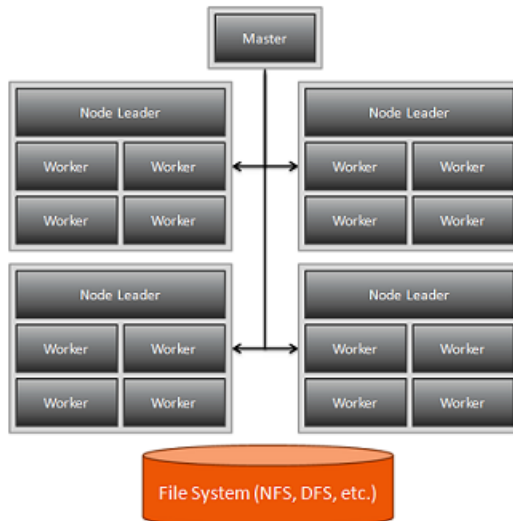


Everything is installed locally. The master is local and also functions as the node leader. All workers are installed locally. The number of workers can be configured based on the local computer's specifications. The computer that hosts the master will also host IDL and ENVI.

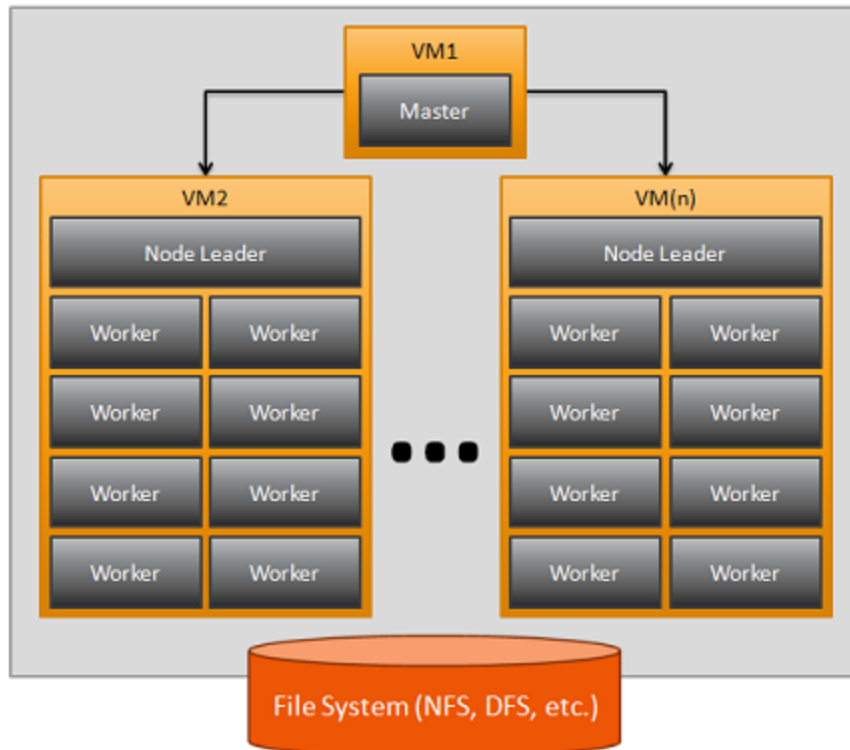
A drawback of having the master and workers on the same node is that workers could consume all of the computer's resources and overwhelm the master in a production environment. If workers are using all of the resources, then the master will have trouble communicating with clients or other workers. Examples of processing that will consume all resources are ENVI native processing, Feature Extraction, and IDL thread pool.

Cluster of Computers

A *cluster* is a set of networked machines that have one common "master" machine, contain a common working directory, and share a common job database and file system. The following diagram shows a simple example:



A common configuration is to use Virtual Machines:



On a cluster, the master process is installed on one master machine, which you can specify in the ESE configuration file. An ESE process assumes the role of master if it is the first instance of the process that is started on the machine designated the master.

Each additional machine in the cluster hosts one node leader and multiple workers. A machine is part of the cluster if it is running an instance of the ESE executable that is wired to communicate with the same Master as other machines on the cluster. On any of these machines, the first instance of a process is run as a node leader. It will monitor a configurable number of worker processes. This number can be specified in the ESE configuration file but typically defaults to the number of central processing units (CPUs) on the system. Node leaders, including the Master, will relaunch workers that have exited. (See Knowledge Base for more information.)

An installation of IDL and ENVI also reside on each of these additional machines. Workers can access shared files (ESE files, data, and images) on a distributed file system (DFS) or network file system (NFS).

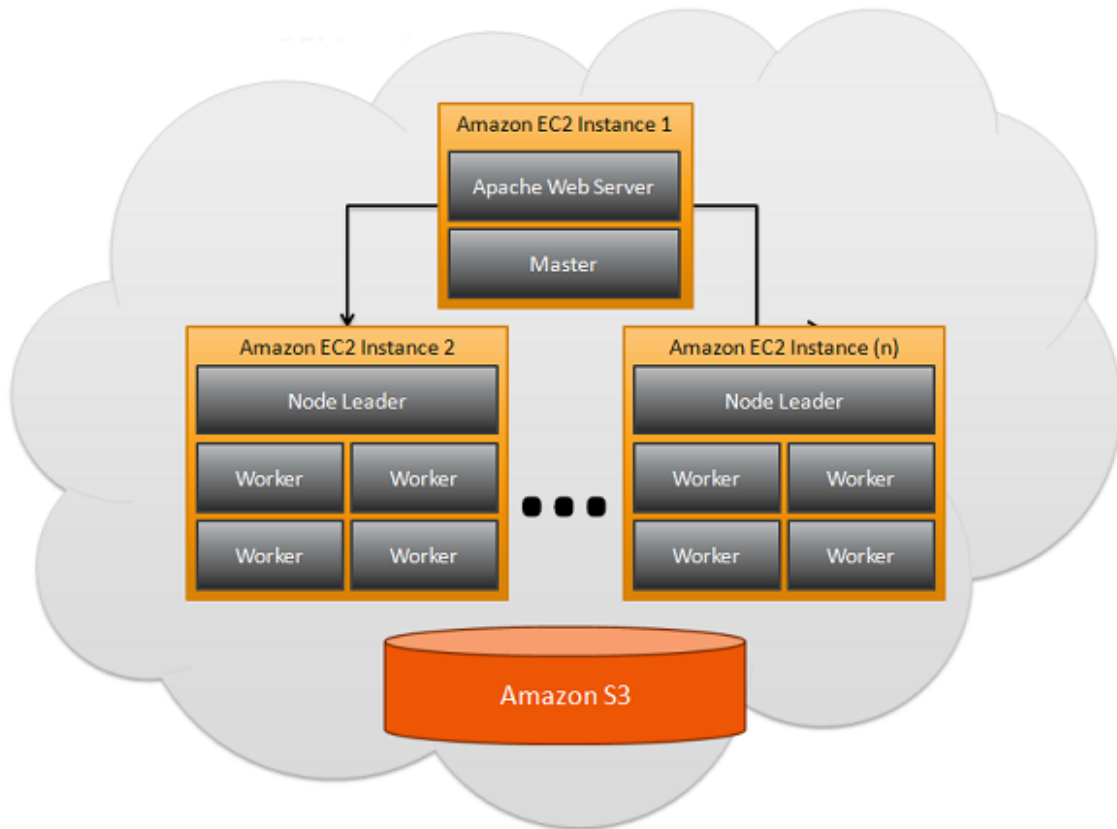
In a cluster of computers or virtual machines (VMs), you need a shared file system for the installation files and task sandboxes. Shared files include:

- `server.cfg`: configuration file
- `docroot` directory: Admin console, job sandboxes, examples, etc.
- Job database
- System-wide logs
- Uploaded tasks

Optionally, you could install one instance of IDL and ENVI on a shared location on the network for all workers to access; however, network speed and traffic can become a bottleneck to processing ESE requests. Ideally, you should install an instance of IDL and ENVI on each node for increased performance.

In larger production environments, an enterprise-level computer may be used to serve as the master. This is typically a single, powerful machine with a good CPU, disk input/output, and network performance.

ESE can also be installed on Web-based cloud services such as Amazon EC2 Web Services. In the following sample configuration, one Amazon EC2 instance hosts an Apache Web Server and the ESE master process. The web server provides access, security, and routing capabilities to the ESE master.



Additional instances of EC2 would each contain a node leader and multiple workers, the number of workers being determined by the specifications of the EC2 instance.

Scaling and Load Balancing

Scaling refers to the ability to grow the number of machines and processes in a clustered environment. Scaling is typically done by turning on/off individual machines. If loads or processing requirements change, you can add or remove node leaders.

ESE can easily scale and balance workloads because it implements a REST architecture and is designed to run on a cluster. As requests arrive from any number of clients, the master process' web server will accept and route the request. If the request is for information about a task (or the system in general), then the master will return the requested information to the client. If the request is to perform a task, then the master will give the request to one of its workers for processing.

Typically, all nodes in a cluster will run from the same installation. The installation can reside on a Network File System (NFS) or Distributed File System (DFS) mount point. However, nodes can have local installations as long as all nodes in the cluster share a common "working directory."

ESE is designed so that any machine on the cluster can execute any incoming task. Additionally, any worker can process any incoming request so the master is free to distribute work for maximum efficiency. As part of this process, workers that have crashed are automatically restarted, minimizing the need for manual intervention by the system administrator.

You can check load balancing from the Admin Console's Nodes tab. In the "Processing Nodes in Cluster" section, check the **Jobs Running** and **Jobs Completed** for each node.

Installation

This section provides details on ESE installation and licensing. Please refer to the Help Articles on the Exelis VIS web site for the most updated installation and licensing instructions.

See the following topics:

- Before You Begin
- Windows Installation Instructions
- Linux Installation Instructions
- Installation Components
- Licensing Overview

Before You Begin

ESE runs on Windows and Linux 64-bit systems:

Platform	Operating System	Hardware	Supported Versions
Windows		Intel/AMD 64-bit	7, 8, Server 2008, 10
UNIX	Linux	Intel/AMD 64-bit	Kernel 2.6.32, glibc 2.12, GTK+ 2.4.13

The Admin Console works with Google Chrome and Mozilla Firefox browsers. The ESE Help requires an HTML5-compatible browser.

Before installing ESE, here are some questions to consider:

- Will you install ESE on a single machine or a cluster? If you install on a cluster, the ESE_WORK_DIR will need to be accessible to all machines.
- How does ESE fit into the overall system or network?
- What other software is going to run on the system where ESE resides?
- Do you need to limit ESE so it doesn't use the entire node or cluster?
- Will ESE be the only process running on the nodes? This is important for picking the numbers of workers, etc.
- Where is the data located? Ideally, the data should be local to the processing ("bring processing to the data") to minimize network traffic and maximize performance.
- What kind of processing will the system need to run? Will you need to optimize your disks before installing ESE?
- Who will be using the system, and what user account will run ESE? Make sure to have the ESE user account set up before attempting installation. The user account should be set up with its own password; a blank password or no password will not work with ESE.

Finally, you should locate a suitable disk for the ESE_WORK_DIR folder structure. Ideally, this should be a disk with high read and write speeds.

Windows Considerations

Windows needs a local installation of IDL and ENVI on each worker node.

ENVI LiDAR is only supported on Windows platforms.

ENVI for ArcGIS® - Services Edition (EFASE) is only supported on Windows 64-bit platforms.

If ArcGIS for Server software is installed on a separate system from ESE, follow the instructions in ArcServer Host and ESE on Different Systems.

Linux Considerations

One installation of IDL and ENVI is sufficient for the entire cluster, if you are using a clustered environment.

You should decide whether to install ESE as an application or daemon. See the Linux Installation Instructions for details.

Windows Installation Instructions

You must have administrator privileges to install ESE on Windows.

1. Insert the installation disk in the appropriate drive. The autorun program should start automatically. If the autorun program does not start, select **Start > Run**. In the Run dialog, click **Browse** to navigate to the drive, select `autorun.exe`, and click **OK**.
2. Click the **Install** button. If you do not already have IDL and ENVI installed, you will be prompted to install them.
3. Select the **ENVI for ArcGIS - Services Edition** option to install ESE along with ENVI for ArcGIS - Services Edition (EFASE). This is for Windows 64-bit systems only. Refer to the EFASE Installation Guide PDF (included with the installation disk or web download) for further details. Otherwise, select the **ENVI Services Engine** option to only install ESE.
4. Choose whether or not to install ESE as a *service* so that it will run as a background process that performs without any user interaction:
 - **Yes, setup the service to start automatically:** Select this option to automatically start the ESE service after a system reboot. This option is convenient for clustered environments with numerous machines.
 - **Yes, setup the service to start manually:** Select this option to manually control when the ESE service runs. See Stop and Start the System for instructions on manually stopping and starting the ESE service after installation.
 - **No:** Choose this option if you will run ESE as an application rather than as a service. See Start ESE As An Application for instructions after installation is complete.
5. If you selected to start the service automatically or manually, you are prompted to enter a **User name** and **Password**. The user account must have a password; a blank password or no password will not work with ESE. Click the **New User Information** button to establish a new user account if needed. Install as a user with restricted privileges because this type of installation poses security hazards. This is because ESE runs with the same permissions as the specified user, including the ability to delete and modify directories and files.
6. Choose the work directory location. The default folder is `C:\Exelis`. Click **Next**.
7. When the installation is complete, you can optionally run the License Wizard. If you choose not to run the License Wizard now, click **No**.

- To run the License Wizard after installation, select **Start > Programs > ENVI x.x > Tools > License Wizard**. Follow the screens to point to a valid license server or to install a license file.
- You can also copy the `license.dat` file to the folder on your machine that contains the installation files for IDL and/or ENVI. The default location is `C:\Program Files\Exelis\License`.
- See Licensing Overview for a description of different license types in ESE.

Start ESE as an Application

You may want to run ESE as an application if you need to debug its installation or are developing new tasks for the server. When running as an application, the console window displays the error messages directly which can help during the debugging process. However, the same error messages are available via the log files as well.

If you would like to start ESE as an application on Windows, follow these steps:

1. Open a command prompt by navigating to: **Start > All Programs > Accessories > Command Prompt** or type `cmd.exe` from the Start menu's search text box.
2. In the Command Prompt window, type: `cd C:\Program Files\Exelis\IDLxx\bin\bin.x86_64` and press the **Enter** key.
3. Then type: `EnviServicesEngine.exe -d`

ESE should now be running as an application.

To stop ESE as an application, type **Ctrl+C** at the command line.

Linux Installation Instructions

You can install ESE on a local machine, on a network file system (NFS), or a distributed file system (DFS).

If you want to try out ESE to see how it works, or to author and test tasks, consider installing on a local machine, not as `root`. See *Install ESE as an Application* for instructions. You can install locally without `root` privileges. Non-`root` users do not have service logon rights, so you will need to manually start the service.

If you want to run ESE in a production environment, then consider installing it as a daemon running under a user with limited permissions. See *Install ESE as a Daemon* for instructions.

Install ESE as an Application

The installer will ask you to specify a location to install ENVI+IDL and a writeable location for ESE work files, such as the job database, job sandbox directories, uploaded tasks, etc.

1. Log in to your system with the user account under which you want to run ESE. For personal use this can be your regular user account.
2. Mount the ESE installation DVD.
3. Enter the following command at your shell prompt, where DVD-PATH is the path to your DVD drive:

```
/bin/sh /DVD-PATH/install.sh
```
4. Follow the prompts to install the software.
5. When the installation is complete, you can optionally run the License Wizard. If you choose not to run the License Wizard now, click **No**.
 - To run the License Wizard after installation, go to the installation path and typing `exelislicense` at the shell prompt. The default location is `/usr/local/exelis/bin`.
 - See *Licensing Overview* for a description of different license types in ESE.

Start ESE as an Application

Enter the following commands at a shell prompt to source and start ESE as an application. The `<nn>` refers to the version of ESE:

```
% cd /se<nn>/bin
% source ese_setup
% EnviServicesEngine
```

ESE will not start if it is not properly licensed. Check the `/se<nn>/logs` folder for details if it does not start.

By default, ESE will create half as many worker processes as there are CPUs on the machine. You can view them in the Admin Console. Open a browser and type in the location:

```
http://<hostname>:<port>/ese/admin
```

Select the Nodes tab to view the nodes and their worker processes.

Install ESE as a Daemon

Installation as a daemon can be on a single machine or on a network or distributed file system. The latter is recommended when deploying the service across multiple machines. Installing ESE in a clustered environment requires `root` privileges during installation, and also a unique user with restricted privileges.

Even though you would install with `root` privileges, we recommend running ESE under a specific user account in order to take advantage of your inherent system-supplied security settings. Note that ESE runs with the same permissions as the specified user, including the ability to delete and modify directories and files. Running ESE under a specific user account allows it to have permission to modify and delete ESE files but not system files.

Another thing to note when installing ESE as `root`: the `root` user has service logon rights. Because of this, if you install ESE as `root`, the daemon can be configured to start automatically as the system boots.

1. Log in to your system as a user with `sudo` privileges.
2. Create a user account that the ESE daemon will run as. All ESE processes will run as this user and hence have its privileges. This should be an account with limited and well-understood privileges. This user account will need write access to an ESE working directory, which by default is `/home/<user>/exelis/se<nn>`.
3. Mount the ESE installation DVD.
4. Enter the following command at your shell prompt, where DVD-PATH is the path to your DVD drive:

```
/bin/sh /DVD-PATH/install.sh
```
5. Follow the prompts to install the software.
6. When the installation is complete, you can optionally run the License Wizard. If you choose not to run the License Wizard now, click **No**.

- To run the License Wizard after installation, go to the installation path and typing `exelislicense` at the shell prompt. The default location is `/usr/local/exelis/bin`.
- See Licensing Overview for a description of different license types in ESE.

Starting the ESE Daemon on Other Nodes

On a cluster of machines, you should start the service on one or more machines besides the master. Issue the following commands on each machine to install or start the daemon and to set up the appropriate service links:

The commands vary between Red Hat and Debian (Ubuntu). Note that `<nn>` is the current, installed version of ESE on your system.

Red Hat:

```
% cd <ESE_WORK_DIR>/exelis/se<nn>/bin
% sudo ./ese-install-initd -u <user>
% sudo /sbin/service ese start
```

Ubuntu:

```
% cd <ESE_WORK_DIR>/exelis/se<nn>/bin
% sudo ./ese-install-initd -u <user>
% sudo /usr/sbin/service ese start
```

The following directories are involved with installation of the daemon:

<code>/usr/local/exelis</code>	Possible ESE installation location.
<code>/etc/init.d</code>	Linux daemons are configured via files and symbolic links in these locations.
<code>/etc/rc.d/rc#.d</code>	

Uninstalling ESE from Linux RedHat Systems

This section pertains to RedHat, Fedora, CentOS, and similar Linux operating systems. To uninstall the daemon, issue the following commands.

1. First, stop the daemon:


```
% sudo /sbin/service ese stop
```
2. Remove the links:


```
% sudo /sbin/chkconfig --del ese
```

For RedHat, use:

```
% sudo /sbin/chkconfig --delete ese
```

3. Remove ESE daemon script from the init.d directory:

```
% sudo rm /etc/init.d/ease
```

After these steps, you can still manually run ESE as an application by running the `EnviServicesEngine` script. The daemon can be set up again by running the `ease-install-initd` script that is in the `<EXELIS_WORK_DIR>/se<nn>/bin` directory (where `<nn>` is the current, installed version of ESE).

To fully uninstall ESE, issue the following commands:

1. Remove the directory where IDL, ENVI and other products are installed:

```
% sudo rm -rf /usr/local/exelis
```

2. Remove the ESE_WORK_DIR:

```
% sudo rm -rf /home/<user>/exelis
```

Note: These steps will delete the jobs database and all tasks that have been uploaded. If you want to keep the tasks, copy the `<EXELIS_WORK_DIR>/se<nn>/tasks` folder to a safe location before uninstalling ESE.

Uninstalling ESE from Debian Linux Systems

These steps pertain to Ubuntu and similar Linux operating systems.

1. First, stop the daemon:

```
% sudo /usr/sbin/service ease stop
```

2. Remove the links.

```
% sudo /usr/sbin/update-rc.d -f ease remove
```

3. Remove ESE from the the init.d directory:

```
% sudo rm /etc/init.d/ease
```

After these steps, you can still manually run ESE as an application by running the `EnviServicesEngine` script. The daemon can be setup again by running the `ease-install-initd` script that is in the `<EXELIS_WORK_DIR>/se<nn>/bin` directory (where `<nn>` is the current, installed version of ESE).

To fully uninstall ESE, issue the following commands:

1. Remove the directory where IDL, ENVI, and other products are installed:

```
% sudo rm -rf /usr/local/exelis
```

2. Remove the ESE_WORK_DIR:

```
% sudo rm -rf /home/<user>/exelis
```

Note: These steps will delete the jobs database and all tasks that have been uploaded. If you want to keep the tasks, copy the `<EXELIS_WORK_DIR>/se<nn>/tasks` folder to a safe location before uninstalling ESE.

Installation Components

This topic presents some basic information about ESE's installed components and their locations on disk. The installation process for ESE creates two separate directories: one holds the IDL and ENVI installations as well as the ENVI Services Engine executable; the other directory contains the `ESE_WORK_DIR` and related files.

IDL and ENVI Directories

The IDL/ENVI directory structure contains the program, help, and example files for IDL and ENVI, as well as the license directory for all Exelis VIS products including IDL, ENVI, and ESE. Additionally, it contains the ESE executable, which is necessary to start ESE as an application. The permissions on this directory structure may be restricted with no write access.

The location of the IDL/ENVI directory depends on the operating system:

Linux: You can choose a new location during installation, but the default location is `/usr/local/exelis`.

Windows: Chosen during installation, but the default location is `C:\Program Files\Exelis`.

ENVI Services Engine Executable

By default, the ESE executable resides in the following folder:

- **Linux (script):** `<IDL_INSTALL_DIR>/bin`. The script is `EnviServicesEngine`. It sets up the environment before calling the executable.
- **Linux (executable):** `<IDL_INSTALL_DIR>/bin/bin.linux.x86_64`. The executable is `EnviServicesEngine`.
- **Windows:** `<IDL_INSTALL_DIR>\idlxx\bin\bin.x86_64`. The executable is `EnviServicesEngine.exe`.

ESE_WORK_DIR

The `ESE_WORK_DIR` is a user-writable directory structure and contains the ESE support files, help and example files, and the `bin`, `docroot`, `logs`, `lib`, and `tasks` directories. The installation process places the `ESE_WORK_DIR` in a separate location from the ESE, IDL, and ENVI executables.

Linux Clustered Environment: Both the IDL and ENVI directories, as well as `<ESE_WORK_DIR>` should be on a network location. If a drive letter is necessary, creating a directory symbolic link will also work. See Knowledge Base for more information.

Windows Clustered Environment: Choose a location on a network or shared drive using a UNC syntax.

The following directories are installed below `ESE_WORK_DIR`:

Bin

This directory contains the ESE configuration file (`server.cfg`), scripts for setting up ESE, as well as Linux scripts to run ESE as a daemon.

Docroot

This directory contains:

- The web server's directories
- The Admin Console web application files
- The ENVI 3D Web Viewer web application files
- Job sandboxes, which are directories where a worker process writes IDL/ENVI files (by default). It will also generate a JSON file for the HTTP request, write an IDL log file, and place a temporary directory here for IDL. Each job gets its own sandbox within the `ese/jobs` directory. Each sandbox is named according to the job number. They are cleaned up by the ESE master process after a period of time.
- The configuration files needed for ENVI® for ArcGIS® - Services Edition.

The `docroot/ese/data` folder contains sample data files that can be used to test ENVI analytics. The `docroot/ese/data_upload` directory contains data uploaded by client applications. Use the Data tab in the Admin Console to see the available data files. The `data` directory is the recommended "safe" directory for adding sample data files. For example, the job sandbox subdirectories in the `jobs` directory also have their contents served up, but ESE's cleanup process will eventually delete them as specified in the configuration file.

Job Database (jobdb file)

ESE employs a small SQLite database to store information relating to the execution of jobs:

- Job queue, start, and finish times
- Job status information

You should not change the Job Database; however, it is available for querying if you need to extract detailed job statistics. If so, the best option is to query it when the system is offline as the SQLite database does not allow for concurrent users.

Logs

ESE maintains a series of log files that you can use for troubleshooting and maintenance. For example, if your system doesn't boot up correctly or if you have errors with uploading tasks, you can check the log files through the Admin Console or on disk.

Licensing Overview

Exelis VIS licenses each ESE system individually. A system consists of one designated master process and a number of worker processes. These processes may either be installed on one machine locally or across several machines in a networked environment.

We base the license level of your system on two components that work together:

1. The chosen license Feature Group *and*
2. The number of concurrent workers purchased for the system.

Feature Groups

Exelis VIS offers three basic ESE Feature Groups:

Feature Group	Components Included
1	ESE + IDL
2	ESE + IDL + ENVI + FX* + NITF [†]
3	ESE + IDL + ENVI + FX* + NITF [†] + LiDAR

* ENVI Feature Extraction

[†] National Imagery Transmission Format extension

Additional components, such as DEM Extraction and DICOM Network Services, are available and will be licensed separately. Check with an Exelis VIS sales or professional services representative for more information on obtaining add-on components.

Concurrent Workers

The number of concurrent worker licenses purchased is independent of the number of actual workers available on the system. A general rule-of-thumb is to have the same or more actual workers available as the number of concurrent worker licenses purchased.

Job Processing and Node Locking

As job requests come in to the master, it assigns them to workers until the system reaches the maximum number of allowed concurrent workers actively processing jobs. When the system reaches that maximum number, the master queues jobs until a worker "license" is freed upon the completion of a task. The master then sends out the next job in the queue for processing.

If an in-process job requires additional features not licensed on your system, the job will fail. Check your server logs for details.

Configuration Options

This section describes various configuration options, typically performed by a system administrator.

- Configuration File
- IDL Reset Options
- IDL Thread Pool

Configuration File

The configuration file (`server.cfg`) contains settings for the ESE installation and environment. This file also defines the ESE cluster if more than one machine is used; it should be accessible to all machines that comprise the cluster.

By default, the configuration file installs in the following location:

- **Linux:** `<ESE_WORK_DIR>/bin/server.cfg`
- **Windows:** `<drive>:\Exelis\se<nn>\bin\server.cfg`

Where `<nn>` is the current, installed version of ESE on your system.

Back up your original configuration file before you make changes so that you can refer to the default settings if needed.

ESE accesses the `server.cfg` file only on initial startup of every master and worker process.

Keys

The following table lists the keys in the configuration file, their default values, and a brief explanation for each one.

Key	Default Value	Description
<code>master</code>	<code><machine_name></code>	The name of the master machine. Optionally use the full name or IP address to avoid problems.
<code>version</code>	<code><version></code>	Must match the value of the current ESE version.
<code>ese_work_dir</code>	<code>{ese_work_dir}</code>	The working directory for ESE on the file system. It could be a local disk for a non-clustered environment or a shared NFS mount or a distributed file system (DFS) for a clustered environment. By default, this field is already set in the startup script by the <code>ESE_WORK_DIR</code> environment variable on Linux, or by a registry setting on Windows.
<code>document_root</code>	<code>\${ese_work_dir}/docroot</code>	Web server's documentation root.

Key	Default Value	Description
logs_root	<code>\${ese_work_dir}/logs</code>	Location of ESE system log files.
tasks_root	<code>\${ese_work_dir}/tasks</code>	Location of tasks that have been uploaded to ESE.
pro_code_root	<code>\${ese_work_dir}/lib</code>	The location of the <code>/lib</code> directory. You can place PRO code and DLLs here for all tasks to access.
job_database	<code>\${ese_work_dir}/jobdb</code>	SQLite database of jobs.
config_root	<code>\${ese_work_dir}/bin</code>	Not required; only used by distributed file systems
using_envi	yes	If the server will be using ENVI functionality, set this to <code>yes</code> .
reset_after_each_task	closeEnvi	Defines how IDL/ENVI will be managed for each job. See IDL Reset Options for more information.
listening_ports	8181	Defines the server port. You may need to change the value of this key for firewall security.
enable_directory_listing	yes	Controls serving directory listings for folders without an <code>index.html</code> , making files undiscoverable. An effect is whether the Data tab of the admin console should list <code>docroot/ese/data</code> contents.
purge_items_older_than	2 days	Defines the age at which log files and job folders are deleted. Units are days, hours, or minutes.
synch_execution_wait_time	10	Defines the number of seconds the server will wait for a synchronous job to complete. Once a synchronous execution request is sent to the server, the server will wait <i>n</i> seconds for the ENVI Services Engine system to compute a response for return to the client. If it takes longer than

Key	Default Value	Description
max_runtime	unlimited	this amount of time, the server will return a 504 Gateway Timeout to the client. Sets the maximum number of seconds the server will wait for a job to finish. All jobs that exceed this time limit will be canceled*. A value less than or equal to zero signifies no limit (i.e., jobs will not be canceled due to running time).
workers_on_master_node	half_of_cpu_count	Defines the number of worker processes that run on the master's machine. Available values are: half_of_cpu_count, same_as_cpu_count, or an integer ≥ 0.
workers_on_worker_nodes	same_as_cpu_count	Defines the number of worker processes that run on machines other than the master. Available values are: half_of_cpu_count, same_as_cpu_count, or an integer ≥ 0. If there is only one node (the master) then ESE does not use this parameter.
num_idl_threads	-1	Defines the number of threads (and therefore CPUs) IDL will use on the machine. Defaults to "-1" which means "not set." For additional information, see the topic "Controlling the IDL Thread Pool" in the IDL Help documentation.
num_tbb_threads	-1	Defines the number of TBBs (Threading Building Blocks) that will be used on the machine. Defaults to "-1" which means "not set."

** When ESE cancels the job, it is actually "killing" the worker process that was running the job. This causes the worker to re-enter a pristine state by killing the operating system's process and starting a new one.*

Version History

1.0	Introduced
1.0.1	Remove references to GPSync, GPASync, the service descriptions, and the catalog name. This functionality has been moved to the <code>catalog.cfg</code> file which is automatically generated at runtime and edited through the Admin Console.
5.1	Obsoleted two fields: <code>purge_database_records_older_than</code> and <code>authentication_domain</code> Renamed <code>engines_on_master_node</code> to <code>workers_on_master_node</code> Renamed <code>engines_on_worker_nodes</code> to <code>workers_on_worker_nodes</code> Changed the default value of <code>workers_on_master_node</code> to <code>half_of_cpu_count</code> Added <code>num_idl_threads</code> and <code>num_tbb_threads</code>
5.2	Renamed <code>distributed_fs</code> field to <code>envi_work_dir</code> . Removed the <code>data_root</code> and <code>num_threads</code> fields.

IDL Reset Options

You can specify when and how to reset IDL in between each ESE task invocation. Indicate the reset option globally at the server level, or locally at the task level.

Global Reset

Set the `reset_after_each_task` value to `yes` in the `server.cfg` file, for example:

```
# Should IDL reset after each job is run?
reset_after_each_task = yes
```

Task Reset in the *.task file

You can also choose to reset IDL at the task level, independent of the global setting. In the `*.task` file, change the `taskReset` option, for example:

```
{
  "name":"test_taskreset_yes",
  "displayName":"test_taskreset_yes",
  "executionType":"synchronous",
  "taskReset":"yes",
  "serverVersion":"5.2",
  "parameters":[]
}
```

Valid Values

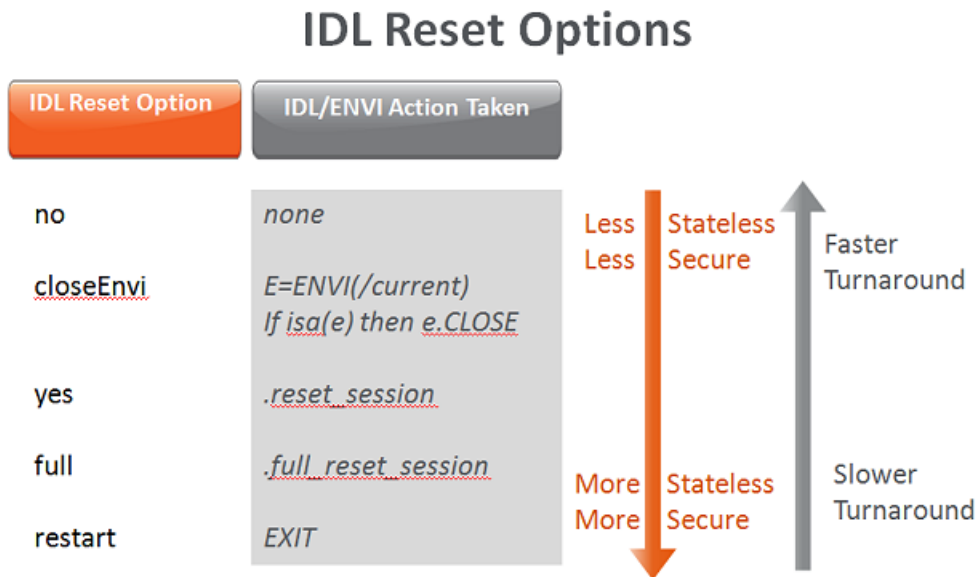
The valid values for the task reset option are the same at both the global and local levels:

- `no`
- `closeEnvi`
- `yes`
- `full`
- `restart`

When the values for the reset option differ at the global and local level, the more secure/stateless option wins. See the diagram below for additional information. For example, if your global reset setting is `closeEnvi` but the task's reset option is set to `full`, then a full reset will occur when that particular task runs.

If a task uses a DLM, you should set the `restart` option to make sure native functions are not cached.

The options and their advantages/disadvantages are summarized in the graphic below.



Additionally, the table below describes each option in detail. If `using_envi` is set to `yes` in the ESE configuration file, ENVI will reinitialize immediately after the job completes, so the performance cost will be incurred at the end of the current job instead of the beginning of the next.

IDL Reset Option	Comments
no	No reset command is invoked. Input and output variables that are defined for the task are cleaned up. Other variables that may have been created at <code>\$MAIN\$</code> by the task will leak.
closeEnvi	Shuts down ENVI if it is running. This keeps the ENVI code in memory, so reinitialization time is about half that of a fresh start. This option removes any data that might be in the data collection and closes any remote connections.
yes	Resets the IDL session: All variables will be cleaned up and all IDL PRO code will be unloaded. Refer to the IDL <code>.RESET_SESSION</code> description.

IDL Reset Option	Comments
full	Fully resets the IDL session, which cleans up all variables and unloads all IDL PRO code and shared libraries. Refer to the IDL .FULL_RESET_SESSION description for details.
restart	Completely exits the IDL process. This has the effect of shutting down the ESE worker process. A new worker will be automatically started.

IDL Thread Pool

If you do not want ESE to use all of a system's resources, then you will need to understand IDL's Thread Pool. This is an IDL system preference available from the IDL Workbench or the Command Line. From the IDL Workbench menu, select **Window > Preferences > IDL > Interpreter** to access the preferences for the thread pool.

System Administrators need to hard-code the maximum number of cores that IDL can use.

Many of the IDL routines that use the IDL thread pool also have thread pool keywords. Those writing ESE tasks should use the thread pool keywords whenever possible.

More information on the Thread Pool is available in the IDL Online Help in these topics:

- The IDL Thread Pool
- Controlling the IDL Thread Pool
- Routines that Use the Thread Pool
- Thread Pool Keywords

System Administration

This section is designed for system administrators who manage tasks, services, jobs, nodes, and system security. You will perform all system administrative tasks using the web-based Administration Console, or *Admin Console*.

To open the Admin Console:

- Access it from your desktop, or
- Open a web browser and enter the following address:
`http://hostname:port/ease/admin`, where 8181 is the default port value.

See the following sections related to system administration:

- Stop and Start the System
- Manage the Catalog
- Manage Tasks
- Monitor Jobs
- Check Log Files
- Monitor Usage Statistics
- Security
- Command Line Options

Stop and Start the System

This topic describes how to manually stop and start the ESE service if you selected to install ESE as a service, not as an application. When you stop and restart the service, any jobs that were in a "Queued" status will automatically start again. However, you can lose jobs that are running; they will not restart when ESE starts again.

If you selected to setup the service to start automatically during the installation process, the service should start automatically after a machine restart.

The Admin Console cannot be used to stop or start the ESE services. See the steps below.

Manually Stop the Service

Windows

1. Open the Services Manager by selecting **Start > Control Panel > System and Security > Administrative Tools > Services**.
2. Highlight **ENVI Services Engine Service**, and click the **Stop** link on the left.

Linux

- To stop the daemon on RedHat systems, issue the following command:

```
% sudo /sbin/service ese stop
```
- To stop the daemon on Debian systems, issue the following command:

```
% sudo /usr/sbin/service ese stop
```

Manually Start the Service

Any time you restart ESE, you must start the master node first before attempting to start any of the worker nodes. The master node maintains the licensing for the workers, so the workers must be able to communicate with the master in order for the license transfer to take place.

Note: You should ensure that all ESE processes are stopped before restarting ESE.

Windows

1. Open the Services Manager by selecting **Start > Control Panel > System and Security > Administrative Tools > Services**.
2. Highlight **ENVI Services Engine Service**, and click the **Start** link on the left.

Linux

- To start the daemon on RedHat systems, issue the following command:

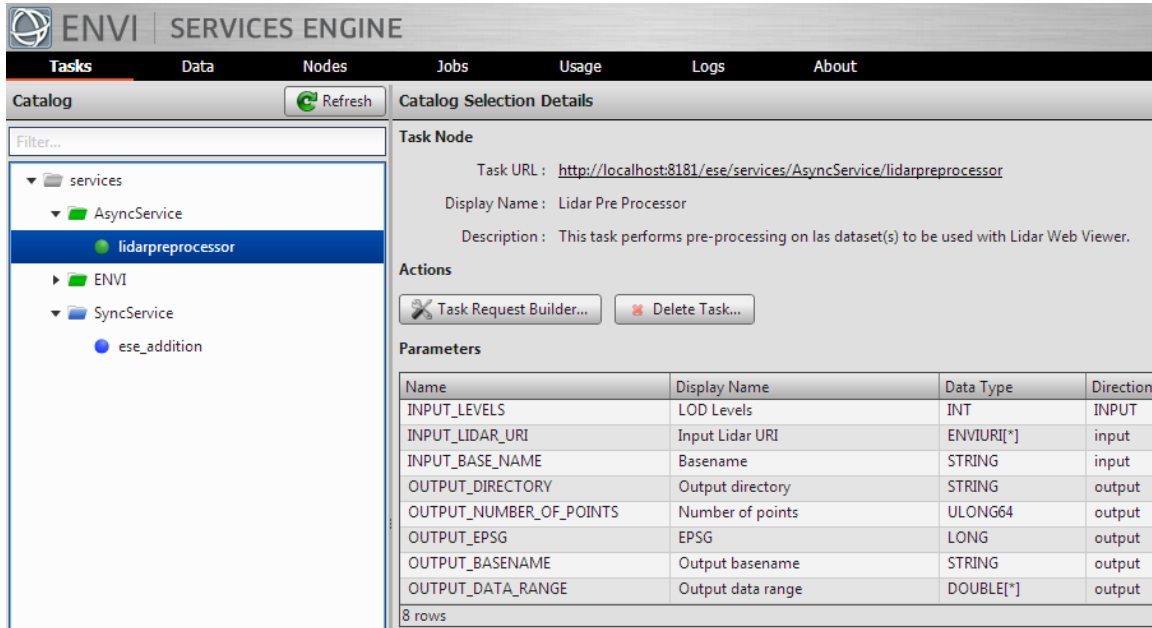
```
% sudo /sbin/service ese start
```

- To start the daemon on Debian systems, issue the following command:

```
% sudo /usr/sbin/service ese start
```

Manage the Catalog

Click the Tasks tab in the Admin Console to see a list of folders, services, and task names. These are displayed under the Catalog section on the left.



Task Node

Task URL : <http://localhost:8181/ease/services/AsyncService/lidarpreprocessor>

Display Name : Lidar Pre Processor

Description : This task performs pre-processing on las dataset(s) to be used with Lidar Web Viewer.

Actions

[Task Request Builder...](#) [Delete Task...](#)

Parameters

Name	Display Name	Data Type	Direction
INPUT_LEVELS	LOD Levels	INT	INPUT
INPUT_LIDAR_URI	Input Lidar URI	ENVIURI[*]	input
INPUT_BASE_NAME	Basename	STRING	input
OUTPUT_DIRECTORY	Output directory	STRING	output
OUTPUT_NUMBER_OF_POINTS	Number of points	ULONG64	output
OUTPUT_EPSG	EPSG	LONG	output
OUTPUT_BASENAME	Output basename	STRING	output
OUTPUT_DATA_RANGE	Output data range	DOUBLE[*]	output

8 rows

Three services are provided with your ESE installation:

- **AsyncService:** This folder contains *asynchronous* tasks, which do not block the client during execution. You must periodically check the job status for completion. Processing results are embedded in the JSON string when the job is finished.
- **ENVI:** A group of asynchronous ENVI data-processing tasks, called *ENVITasks*. You can run these the same way as you would ESE tasks. For more information on ENVITasks, please see the ENVI Help installed with your copy of ENVI.
- **SyncService:** This folder contains *synchronous* tasks, which block the client until they are finished running. These are typically short tasks. Use the `synch_execution_wait_time` field in the ESE configuration file to set the number of seconds in which the server will wait for a synchronous job to complete.

Selecting the name of a folder or service in the Catalog will display a detailed description of it in the right pane. Fields include URL, Description, Execution Type, and the number of tasks it contains. Selecting a task name displays its URL, name, and parameters.

Use the **Filter** text box to display only those tasks whose name contains the filter criteria. For example, if you type "raster" into the **Filter** text box, the Admin Console will only display the tasks that match the term "raster."

Right-Click Menu Options

Right-click on any point in the catalog tree to access a menu of options:

Folder-Level Options

- **New Folder** - Creates a new folder below the currently selected folder. You can only create folders under existing folders. You cannot create a folder under a "service."
- **New Service** - Opens a dialog that allows you to create a new service below the current folder. Enter the name of the service and choose whether this will be a synchronous or an asynchronous service. You can also add an optional description for the new service in this dialog box. Click **OK** to see the new service in the Catalog.
- **Rename Folder** - Opens a dialog that allows you to change the folder's name.
- **Delete Folder** - Deletes the selected folder and all of the services and tasks below it. This also removes any related task files uploaded to disk.

Service-Level Options

- **Upload** - Launches the Upload ESE Task dialog. See Upload (Publish) a Task for more information.
- **Edit Properties** - Opens a dialog that allows you to change the service's Name and Description.
- **Delete Service** - Deletes the selected service and all of the tasks below it. This also removes any related task files uploaded to disk.

Task-Level Options

- **Delete Task:** Deletes the selected task, including any task files uploaded to disk.

Once you have created one or more service endpoints, you can then upload tasks. Note that tasks cannot be uploaded directly to a folder; they must reside under a service.

Move Tasks

You can move tasks from one service to another using drag-and-drop. This works across services within the same folder, or across folders and services. However, you cannot drag-and-drop across execution type - your source and destination services must be of the same "type" (asynchronous or synchronous). You cannot move a synchronous task to an asynchronous service, or vice-versa.

Note: If you change the folder service location, existing clients using those task(s) will break. The client HTML will need to be checked and/or fixed by the writer of the client to look for the task in the new location. You should only change the location during the initial upload and task organization process. Avoid doing this once the client applications are actively using these tasks.

Manage Tasks

The Tasks tab in the Admin Console allows you to manage the catalog of tasks, upload and organize tasks, build task requests, and submit jobs. See the following sections:

- Upload Tasks
- Submit Jobs

Upload Tasks

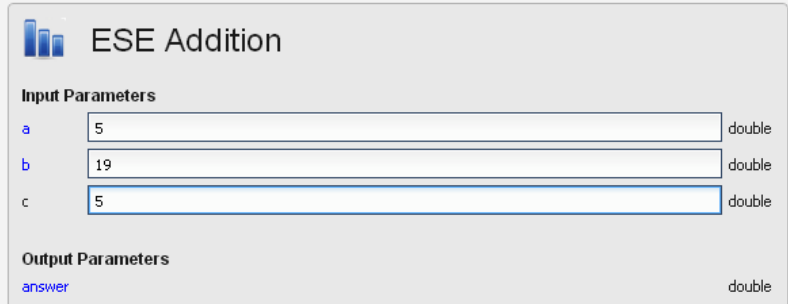
Follow these steps to upload (publish) tasks to ESE. See Writing Tasks for instructions on creating task files.

1. Create a compressed ZIP file with the following:
 - IDL code (`.pro`)
 - Task definition file task (`.task`)
 - DLLs, if used
 - Sample data, if used
 - IDL save files (`.sav`), if used
2. Start the Admin Console.
3. Under the Tasks tab, select either an **AsyncService** or **SyncService** location, depending on the type of task you are uploading. *Asynchronous* tasks do not block the client during execution. You must periodically check the job status for completion. This is the most common type of task. *Synchronous* tasks perform one simple task and block the client until they are finished running.
4. Right-click on the Service name and select **Upload**. Alternatively, click the **Upload** button near the top of the window. ESE will display an Upload Dialog.
5. Click the **Add Files** in the Upload Dialog and navigate to the `.zip` file that you created above. Select it and click **pen** to upload it.

Note: The `ese_addition` task is already installed, so for this example you do not need to create and upload it. Find other examples in `<ESE_WORK_DIR>/docroot/ese/examples`.
6. To delete a task, right-click on the task in the task tree and select **Delete Task**.

Submit Jobs

This section describes how to manually run jobs. When you run a task, it is called a *job*. Under the Data tab, select a task name in the Catalog. Then click the Task Request Builder button. This opens a popup window where you can experiment with input values for the parameters and then run the task. For this example, the Task Request Builder window displays the parameters and fields:



ESE Addition

Input Parameters

a	5	double
b	19	double
c	5	double

Output Parameters

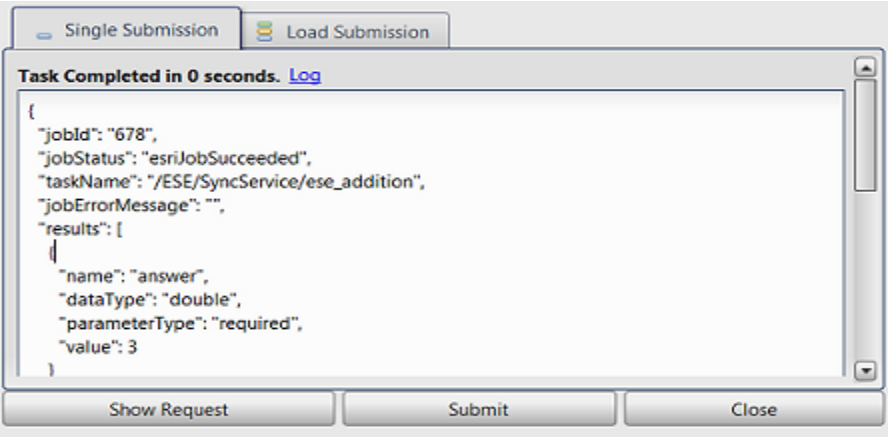
answer	double
--------	--------

When the required parameters have values, the **Show Request** and **Submit** buttons become available at the bottom of the window.

Show Request displays a URL that invokes the task. Use this URL when developing clients that run tasks.

To run the task once, click **Submit**.

The job's response displays in the Single Submission tab in JSON. Scroll to the end of the JSON to see the results. Client applications will need to parse the JSON to extract the result(s).



Single Submission Load Submission

Task Completed in 0 seconds. [Log](#)

```
{
  "jobId": "678",
  "jobStatus": "esriJobSucceeded",
  "taskName": "/ESE/SyncService/esr_addition",
  "jobErrorMessage": "",
  "results": [
    {
      "name": "answer",
      "dataType": "double",
      "parameterType": "required",
      "value": 3
    }
  ]
}
```

Show Request Submit Close

The **Load Submission** tab allows you to submit multiple instances of the task in series. Enter the number of submissions (jobs) to run, then click **Submit**. The window shows the progress of the jobs. The links display the log for each job.

Jobs	Time	Status	Log
1			Log
2			Log
3			Log
4			Log
5			Log
6			Log

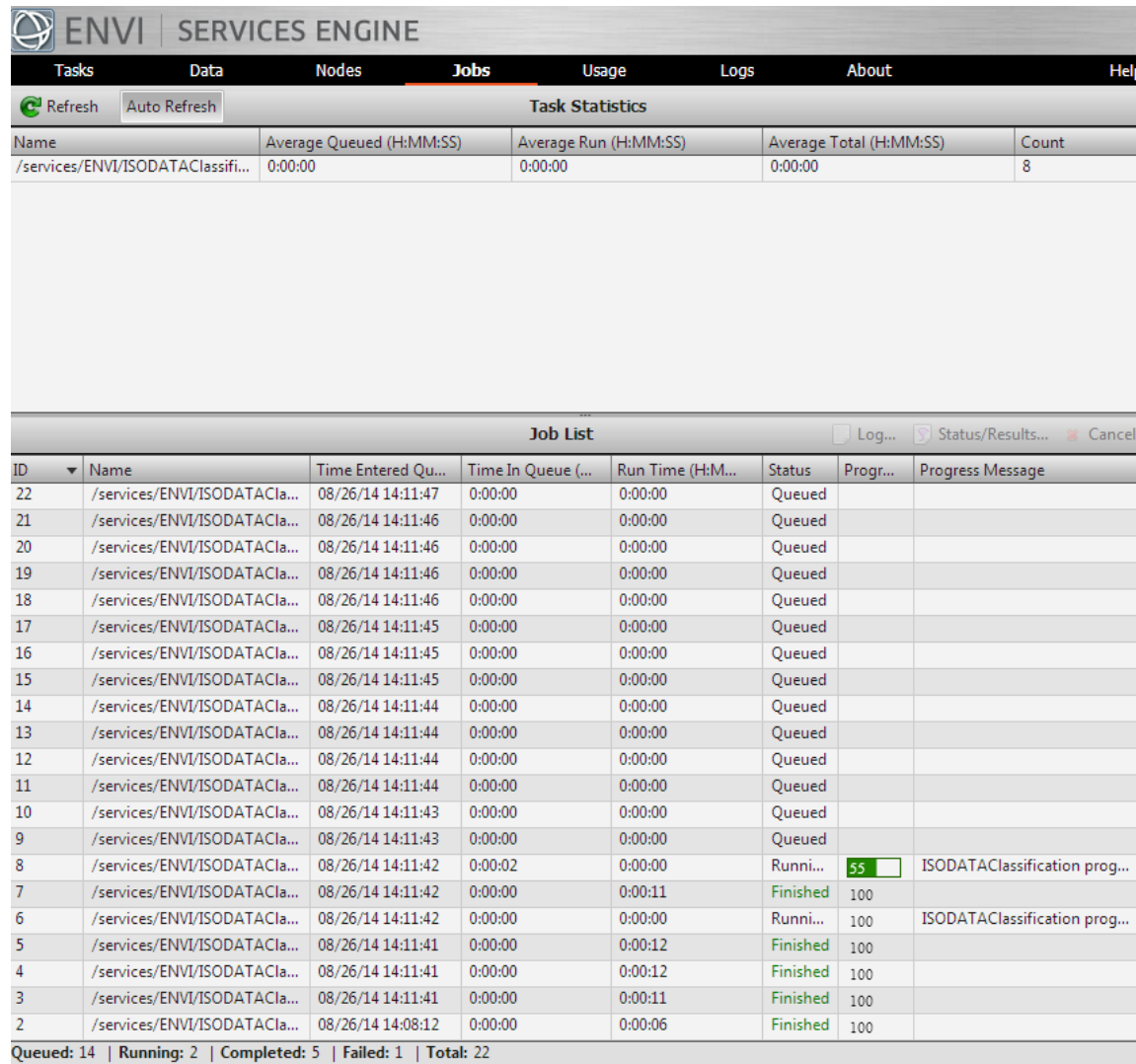
Running a job multiple times works well for Asynchronous Tasks. Running Synchronous Tasks multiple times, or after multiple submissions, requires that you pay attention to the Synch Execution Wait Time (in `server.cfg` file in your installation directory) and how long it will take for the task to run. This is essentially a timeout time. If the synchronous task doesn't complete in this amount of time, it will fail due to timeout.

Monitor Jobs

The Jobs tab allows you to view job statistics and perform basic administrative tasks related to jobs.

See the following section:

- Job Statistics
- Job List
- Check Job Status



The screenshot shows the ENVI SERVICES ENGINE interface with the **Jobs** tab selected. The interface includes a top navigation bar with tabs: Tasks, Data, Nodes, **Jobs**, Usage, Logs, About, and Help. Below the navigation bar, there are buttons for 'Refresh' and 'Auto Refresh'. The main content area is divided into two sections: 'Task Statistics' and 'Job List'.

Task Statistics

Name	Average Queued (H:MM:SS)	Average Run (H:MM:SS)	Average Total (H:MM:SS)	Count
/services/ENVI/ISODATAClassifi...	0:00:00	0:00:00	0:00:00	8

Job List

ID	Name	Time Entered Qu...	Time In Queue (...)	Run Time (H:M...	Status	Progr...	Progress Message
22	/services/ENVI/ISODATACla...	08/26/14 14:11:47	0:00:00	0:00:00	Queued		
21	/services/ENVI/ISODATACla...	08/26/14 14:11:46	0:00:00	0:00:00	Queued		
20	/services/ENVI/ISODATACla...	08/26/14 14:11:46	0:00:00	0:00:00	Queued		
19	/services/ENVI/ISODATACla...	08/26/14 14:11:46	0:00:00	0:00:00	Queued		
18	/services/ENVI/ISODATACla...	08/26/14 14:11:46	0:00:00	0:00:00	Queued		
17	/services/ENVI/ISODATACla...	08/26/14 14:11:45	0:00:00	0:00:00	Queued		
16	/services/ENVI/ISODATACla...	08/26/14 14:11:45	0:00:00	0:00:00	Queued		
15	/services/ENVI/ISODATACla...	08/26/14 14:11:45	0:00:00	0:00:00	Queued		
14	/services/ENVI/ISODATACla...	08/26/14 14:11:44	0:00:00	0:00:00	Queued		
13	/services/ENVI/ISODATACla...	08/26/14 14:11:44	0:00:00	0:00:00	Queued		
12	/services/ENVI/ISODATACla...	08/26/14 14:11:44	0:00:00	0:00:00	Queued		
11	/services/ENVI/ISODATACla...	08/26/14 14:11:44	0:00:00	0:00:00	Queued		
10	/services/ENVI/ISODATACla...	08/26/14 14:11:43	0:00:00	0:00:00	Queued		
9	/services/ENVI/ISODATACla...	08/26/14 14:11:43	0:00:00	0:00:00	Queued		
8	/services/ENVI/ISODATACla...	08/26/14 14:11:42	0:00:02	0:00:00	Runni...	55	ISODATAClassification prog...
7	/services/ENVI/ISODATACla...	08/26/14 14:11:42	0:00:00	0:00:11	Finished	100	
6	/services/ENVI/ISODATACla...	08/26/14 14:11:42	0:00:00	0:00:00	Runni...	100	ISODATAClassification prog...
5	/services/ENVI/ISODATACla...	08/26/14 14:11:41	0:00:00	0:00:12	Finished	100	
4	/services/ENVI/ISODATACla...	08/26/14 14:11:41	0:00:00	0:00:12	Finished	100	
3	/services/ENVI/ISODATACla...	08/26/14 14:11:41	0:00:00	0:00:11	Finished	100	
2	/services/ENVI/ISODATACla...	08/26/14 14:08:12	0:00:00	0:00:06	Finished	100	

Queued: 14 | Running: 2 | Completed: 5 | Failed: 1 | Total: 22

Job Statistics

For each task run, this section of the screen displays summary statistics:

- **Name:** The location and name of the task.
- **Average Queued:** For all jobs of this task that are run, the average amount of time spent in the queue. Check this periodically to see if it is consistently large (greater than the Average Run time). If so, you should consider increasing the number of workers, increasing the number of nodes, or optimizing the worker-to-CPU core ratio.
- **Average Run:** For all successful instances of this task, the average amount of time taken for the jobs to complete. For jobs that fail, ESE counts their run times as zero (00:00:00). Note that jobs that take < 1 sec will also report as zero (00:00:00).
- **Average Total:** Average Queued + Average Run
- **Count:** The number of times this task has been queued AND either completes successfully or fails. Does *not* include jobs where this task was queued but manually canceled before completion.

Additionally, clicking on the heading of the right-most column allows you to choose the columns you want to display, as well as their sizes.

Job List

This section displays a detailed listing of each individual job for the tasks displayed under Job Statistics.

- **ID:** The ESE-assigned job ID for each individual job. This matches the ID in `gp_job.json`.
- **Name:** The location and name of the task.
- **Time Entered Queue:** The date/time the job was queued.
- **Time Started:** The date/time the job started processing.
- **Time Completed:** The date/time the job finished.
- **Time in Queue:** The time this particular job spent in the queue before it starts processing (Time Started - Time Entered Queue).
- **Run Time:** The amount of time taken for the job to complete, if successful (Time Completed - Time Started).
- **Time in Queue + Run Time:** Total elapsed time for this job if it is successful.

- **Status:** The current status of this particular job. Available statuses are: Queued, Running, Finished, Failed, Canceling, and Canceled.
- **Progress:** A value representing the progress of the job, as passed by !SERVER.Notify and as defined in the `gp_job.json` file.
- **Progress Message:** Displays messages as passed by !SERVER.Notify and as defined in the `gp_job.json` file.

Clicking on the heading of the rightmost column allows you to choose the columns you want to display, as well as their sizes. Additionally, most of the fields in the Job List are sortable. Click on the column header to toggle the order of alphabetical sorting.

If you are an Application Developer, use the Job List to check the run time of your jobs to make sure they are reasonable for the amount and type of processing they contain. If you notice problems, check the log files, or contact your system administrator for assistance.

Check Job Status

To check a job status, select a job in the Job List, then click the **Status/Results** button. The system will display the `gp_job.json` contents for this particular job. You can also click the corresponding **Log** button to display the detailed IDL log for a particular job.

Jobs that fail during processing will display a "Failed" message in the Status field of the Job List. To investigate the cause of the failure, check ESE's logs:

- Check the Job Log to try to pinpoint why it failed. Select the job that failed then click the "Log..." button in the Job List's header.
- It may be helpful to check the IDL Log or the Error Log on the Nodes tab. These two logs display information regarding individual nodes used in processing jobs.

If a job's status displays "Running" for too long, the worker may have crashed or stopped/restarted while the job was running. If a worker crashes or is stopped while a task is running, ESE will not restart the task. To cancel queued or active jobs, you can cancel the job by selecting its name and clicking the **Cancel** button. When ESE cancels the job, it is actually killing the worker process that was running the job. This causes the worker to re-enter a pristine state by killing the operating system's process and starting a new one.

Check Log Files

ESE has several log files available to help you monitor errors and performance at the Master, Node, and Job levels. See the following topics for more information:

- Master Logs
- Node Logs
- Job Logs

You can also access the log files directly on the ESE system without going through the Admin Console. ESE stores the system and node-level log files in the `\logs` directory of the `ESE_WORK_DIR`.

We recommend that your System Administrator regularly check these various log files to make sure your ESE system and its jobs are functioning properly.

Master Logs

Click the Logs tab in the Admin Console to access the Error Log. The **Refresh** button at the top of the Logs tab performs an immediate refresh of the screen content of the log files. The **Auto Refresh** button toggles on/off this tab's "auto refresh" capability.

The Error Log displays errors associated with processing the `server.cfg` file:

- Parameter not recognized or is invalid
- Error in a `*.task` file
- Errors with uploading/publishing a task
- ESE Master having difficulties communicating with a node leader

Node Logs

Follow these steps to access logs for each of the Nodes. You should check these logs after starting the ESE service to make sure it started correctly.

1. Open the Admin Console.
2. Click the Nodes tab.
3. Highlight a Node listed in the "Processing Nodes in Cluster" section. The Workers on the selected processing node are displayed at the bottom of the Admin Console.

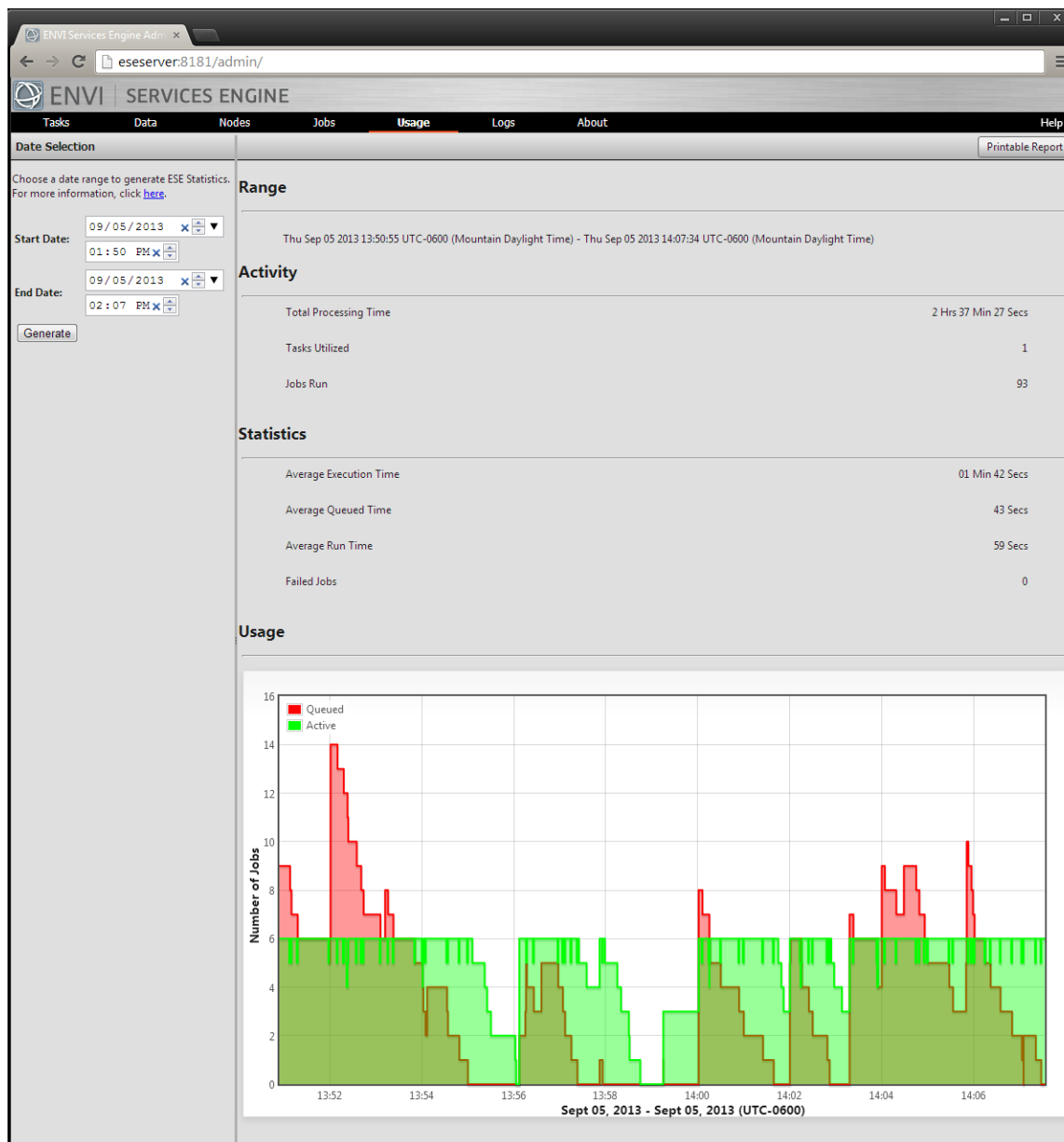
4. Select a Worker, then click a log option:
 - **IDL Log:** Displays the results of IDL processing and any errors associated with the Worker processes.
 - **Error Log:** View errors in processing related to the Master.

Job Logs

Under the Jobs tab of the Admin Console, you can check the log files of any jobs that have failed or stalled. These logs display errors with machine and job initialization. The Job Log displays the processing output for the selected job in a human-readable format. Inside this log, you will first see compilation output, followed by execution messages, and the HTTP address of the output (if any). The end of this log displays the post-job information.

Monitor Usage Statistics

The Usage tab in the Admin Console shows a graphical and statistical representation of the work done by the server:



The various metrics are described as follows:

Range: The date range of the request specified in the time zone of the browser. The date range can be manipulated from the date selection pane on the left side of the screen or by panning/zooming the plot.

Activity: This section provides an overview of the work done by ESE for the requested time period.

- **Total Processing Time:** The sum total of each job's execution time.
- **Tasks Utilized:** The number of unique jobs run on the ESE instance.
- **Jobs Run:** The total number of jobs run on the ESE instance.

Statistics: This section provides a summary of jobs for the requested time period.

- **Average Execution Time:** The average time from request to result.
- **Average Queued Time:** The average time a job had to wait before processing.
- **Average Run Time:** The average time a job took to run once processing started.
- **Failed Jobs:** The number of jobs ESE was unable to complete. Check the error logs in the Admin Console for more information on why a job failed.

Usage: The usage graph gives a visual representation of the load on the ESE instance. The graph displays two lines, a green line which represents the number of jobs processed and a red line which represents the number of job in the queue.

Printable Report: This button will open a new page in the browser that displays the contents of the Usage tab in a printer-friendly but non-interactive HTML page.

Security

System administrators should provide security by way of file and execution permissions, proxies, firewalls, etc. At installation time, you can specify a user name that will be used to run the ESE processes. This user must have permissions appropriate for the systems that ESE will access. In addition, we recommend that you use a secure web server in front of ESE. Any and all security precautions should be considered.

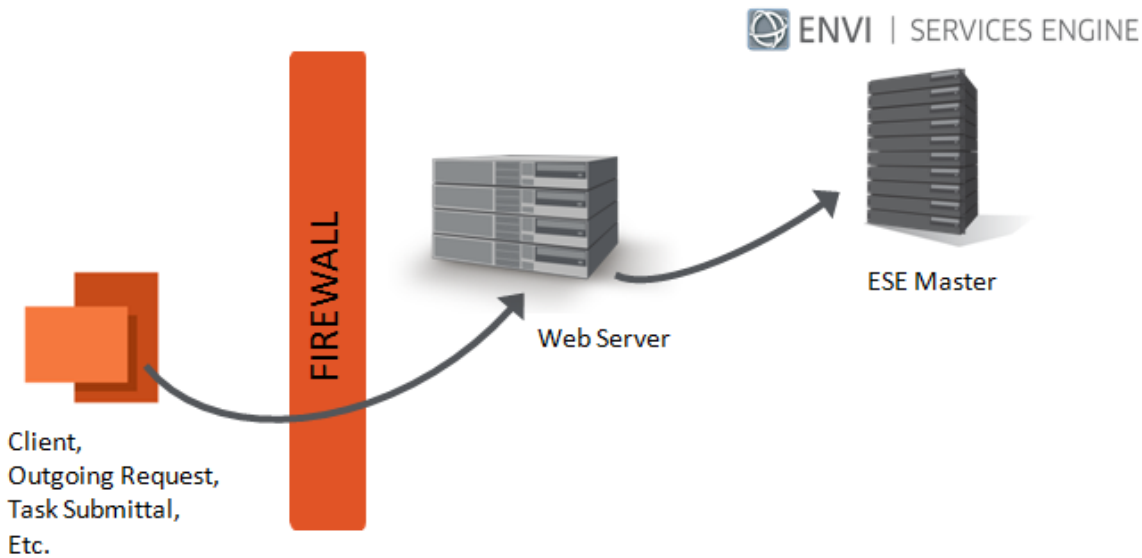
Note: ESE has the same access to any writable file or system as the user running it. The user account responsible for running ESE should have sufficient restrictions in place to protect vital system resources.

Security Best Practices

While ESE itself does not provide security, we do recommend some best practices:

- Understand the level of security that you need on your ESE system before installation. See *Installing on Windows* or *Installing on Linux* for more information.
- Limit access for the ESE user to specific areas of the server or network.
- Even if you install ESE as a user with `root` privileges, run ESE as a user with specific, non-`root` access. This allows you to take advantage of user-level system security. See *Installing on Linux* for more information.

Using a Web Server with ESE



Use a Web Server to Route Requests to the ESE Master

One option to limit access to ESE is to install a web server in front (such as Apache). The web server would then route incoming `/ese` requests directly to the ESE Master. The advantages to this approach are:

- Allows you to set up the system so that the ESE user can run jobs and get status via pass-through to the ESE service end point, but cannot upload tasks.
- The ESE user cannot access the Admin Console, and thus administrative functions, from outside of the firewall. The Admin Console becomes a local host only.
- Limits ESE access only to the Geo-Processing Rest endpoints, thus increasing the system's security.

Writing Tasks

This section is designed for application developers who want to write task files to process data in IDL and ENVI. A task is a set of code and files that inform ESE how to perform some type of processing. The recommended process for writing tasks is outlined below.

- Before You Begin
- Write the PRO Code
- Create a Task Definition File
- Publish the Task

Before You Begin

Consider the following questions when designing a task for data processing:

1. Will the task use IDL code only, or a combination of IDL and ENVI application programming interface (API) code? ENVI has a number of data-processing routines that you can "chain" together to create an entire workflow. Refer to the "Programming" section of ENVI Help. To write a task that takes single or multiple LAS datasets and generates a set of auxiliary files that can be used for viewing and processing, see LidarPreProcessor.
2. Will the task use any dynamically loadable modules (DLMs)?
3. What input parameters are needed?
4. What data types will the task use? See Data Types for list of supported types in ESE.
5. What will be the output, and where does it need to be placed? Tasks are discrete, atomic units of processing. All output must be written to disk or otherwise saved outside of the IDL session.

Write the PRO Code

1. Use IDL, and optionally ENVI API code, to write a program that performs the data analysis.
2. Incorporate the !SERVER system variable in the code:
 - Use the !SERVER.Notify procedure to relay IDL messages to the client about the status of a program, or to get information from the server.

- Clients typically use uniform resource identifiers (URIs) to access data, so you will need to convert any file system paths (for example, C:\Data or \\server\data) to URIs. You can do this using the !SERVER.URLToFile function method. See Options for Selecting Input Data for examples of how clients input data.
- Use the !SERVER.Error procedure to pass any error messages back to the client. This automatically restarts the worker regardless of the the reset settings.

See the System Variables topic for details.

3. Test the code in IDL. See Instantiate a Server for instructions on debugging code when you do not have a server running.
4. Optionally convert the PRO file to a SAV file (.sav), using the IDL Workbench.

The following example creates an image using filled polygons, then animates the scene and saves it to an animated image (.gif) file.

```
PRO SmoothSailing, answer
  x = FINDGEN(100)

  y = 20 * SIN(x*2*!PI/25.0) * EXP(-0.01*x)

  ; Draw the sky and sea.

  p = PLOT(x, y, XRange=[0,99], YRange=[-40,100], $
    FILL_LEVEL=-40, $
    AXIS_STYLE=0, MARGIN=0, DIMENSIONS=[500,400], $
    BACKGROUND_COLOR="light sky blue", $
    /FILL_BACKGROUND, FILL_COLOR="sea green", TRANSPARENCY=100)

  ; Draw the sun
  e = ELLIPSE(0.9, 1, FILL_COLOR="yellow", COLOR="yellow")

  ; Determine the points for the boat to travel.
  xx = 0.5*[-22,-19,-12,-7,8,13,18,23,0.5,0.5, $
    13,8,0.5,0.5,8,3,-2,-7,0,0,-7,-12,0,0]

  yy = 2*[3,-0.7,-1,-1.5,-1.5,-0.7,0.5,3,3,5, $
    5,13,13,15,15,20,20,15,15,13,13,5,5,3]

  ; Draw the boat. Give a Z value to put the boat on top.
  p1 = POLYGON(xx,yy,1,/DATA,FILL_COLOR="burlywood", CLIP=0)

  ; Translate p1 using data coordinates,
  ; translate e using device coordinates.
  FOR i=1,99 DO BEGIN & $
```

```

p.SAVE, 'translatemethod_ex.gif', RES=96, /APPEND & $
  pl.TRANSLATE, 1, y[i]-y[i-1], /DATA & $
  e.TRANSLATE, 1,-0.5 & $
ENDFOR

answer = 'translatemethod_ex.gif'
END

```

Create a Task Definition File

Use the IDL Workbench or any text editor to create a task definition file, which must be in JSON format with a `.task` file extension. This file allows ESE to know the input parameters and other information about the task. It also allows any client to dynamically discover the capabilities of the ESE server and the details of individual tasks according to the Esri® GeoProcessing REST specification.

See the Task Definition File topic for detailed information on the JSON parameters used by ESE. Each parameter defined in the IDL PRO code has a corresponding entry under the `"parameters":` section in the task definition file. In the following example, each parameter has a name and a display name (for use in the Admin Console), a data type, direction (in/out), and a parameter type. These parameters map to the input/output parameters in the PRO code example above.

```

[
  {
    "name": "SmoothSailing",
    "displayName": "SmoothSailing",
    "serverVersion": "5.2",
    "task_reset": "no",
    "parameters": [
      {
        "name": "answer",
        "displayName": "answer",
        "dataType": "string",
        "direction": "output",
        "parameterType": "required"
      }
    ]
  }
]

```

The task definition file determines the order that arguments are passed to IDL for processing. The order in which you specify parameters is irrelevant in the client's HTTP request.

Publish the Task

1. Use the Admin Console to publish the task. See Upload Tasks for instructions. You can also upload tasks directly to ESE from the IDL Workbench. See the ESE Tasks topic in IDL Help for more information.
2. Test the task.
3. Use the Task Request Builder to run the job. See Submit Jobs for instructions.

Task Definition File

Each task definition file included must have a *.task extension, although you can name its base filename to any valid string. The task definition file (*.task) was named config.json in ESE versions older than 5.1. Filenames of config.json are still supported. The *.task file should have the following general JSON format at the task definition level:

Task Object Definition:

```
{
  "name": "<taskName>",
  "displayName": "<displayName>",
  "serverVersion": "<serverVersion>",
  "parameters": {parameter1},{parameter2},{etc.}
}
```

Each parameter within the task (i.e., in the <parameterList> above) should be defined within the JSON task definition as follows:

Parameter Object Definition:

```
{
  "name": "<parameterName>",
  "displayName": "<displayName>",
  "dataType": "<dataType>",
  "direction": "<direction>",
  "parameterType": "<parameterType>",
}
```

Parameters within tasks cannot have values larger than 32,768 characters.

Task Object Keys

The keys and values for the JSON task object, defined above, are:

Key	Required/Optional	Description
name	Required	The exact name of the PRO code routine name. No spaces or special characters allowed.
displayName	Required	A human-readable name for the task. Spaces or special characters are allowed.
executionType	Optional	Values are synchronous, asynchronous, esriExecutionTypeSynchronous,

Key	Required/Optional	Description
<code>serverVersion</code>	Required	or <code>esriExecutionTypeAsynchronous</code> . Default is <code>asynchronous</code> . The version of ESE for which the task was designed. Both <code>version</code> and <code>serverVersion</code> are acceptable values.
<code>taskReset</code>	Optional	Defines whether to reset IDL after each task runs. Values are <code>no</code> , <code>closeEnvi</code> , <code>yes</code> , <code>full</code> , or <code>restart</code> . Default is <code>closeENVI</code> . See Reset Options for more information.
<code>invocationType</code>	Optional	Defines whether the routine is invoked with positional parameters or keywords. Default is <code>positional</code> .
<code>parameters</code>	Required	A list of parameter JSON objects. This may be empty.
<code>description</code>	Optional	The task description.

Note: Any key/value pairs are accepted in the JSON configuration files. However, only ESE-specific pairs are validated.

Parameter Object Keys

The parameter object keys and values are described below. Note that these must be in the same order as in the .PRO code since this pairing defines its connection to the task definition file.

Key	Required/Optional	Description
<code>Name</code>	Required	The exact name as used in the PRO code.
<code>displayName</code>	Required	A human-readable name for the parameter. Spaces or special characters are allowed.
<code>dataType</code>	Required	If one of the supported data types, then validated and cast. Otherwise, passed to IDL as is and normal dynamic typing applies.

Key	Required/Optional	Description
		See Data Types for more information.
direction	Required	Values are "input" or "output".
parameterType	Required	Values are "required" or "optional". If you are running ENVI tasks in ESE and the client REST does not set this parameter, the default parameter value will be used.
defaultValue	Optional	The default value of the parameter in the Task Request Builder or other client to parse. Only used in the Task Request Builder and not used in JSON validation.
choiceList	Optional	A list of options in a combo box for input. Only used in the Task Request Builder and not used in JSON validation.
min	Optional	The minimum value for a slider. Only used in the Task Request Builder and not used in JSON validation.
max	Optional	The maximum value for a slider. Only used in the Task Request Builder and not used in JSON validation.
step	Optional	The step value for a slider. Only used in the Task Request Builder and not used in JSON validation.
description	Optional	The parameter description.

Note: Any key/value pairs are accepted in the JSON task definition files. However, only ESE-specific pairs are validated.

Examples

Task JSON object

```
{
  "name": "ese_addition",
  "displayName": "Addition",
  "serverVersion": "5.2",
  "parameters": {parameter1},{parameter2}
```

```
}
```

Parameter JSON object for long integer data type

```
{
  "name": "xVar",
  "displayName": "Input X Value:",
  "dataType": "long",
  "direction": "input",
  "parameterType": "required"
}
```

Parameter JSON object for a string with choices and a default value

```
{
  "name": "sVar",
  "displayName": "The String",
  "dataType": "string",
  "choiceList" : ["Blue", "Green", "Red", "Cyan"],
  "direction": "input",
  "parameterType": "required",
  "defaultValue": "Blue"
}
```

Parameter JSON object for double-precision data type, optional, with default value

```
{
  "name": "double",
  "displayName": "Double Value",
  "dataType": "double",
  "direction": "input",
  "parameterType": "optional",
  "defaultValue" : 2.5
}
```

Parameter JSON object for color

This requires the REST call for the parameter to provide a three-element array that represents the RGB color triplet.

```
{
  "name": "color",
  "displayName": "Choose a Color",
  "dataType": "color",
  "direction": "input",
  "parameterType": "required"
}
```

System Variables

!SERVER

!SERVER is a read-only system variable that you can use inside of a task's PRO code to relay IDL messages to the client about the status of a program, or to get information from the server.

Properties

!SERVER has several properties that you can use in PRO code to point to specific locations on the server.

Note: The properties that point to directories are fully qualified paths to server locations.

DATADIR

Provides the location to the public data directory of the server. Use this property to store files that are common or useful to other programs on the server.

LOCALDIR

Provides the location of the programs sandbox to the user. It should be used to store temp files or files that are not intended for other programs to use.

DOCROOT

Provides the location of the docroot directory of the server.

DATAURL

Provides the URL of the public data directory of the server.

LOCALURL

Provides the URL to the local directory of the current process.

ROOTURL

Provides the URL to the hostname and port (for example, `http://<hostname>:8181`).

HOSTNAME

Provides the name of the host.

TASKDIR

Provides the location to the tasks directory of the server.

PROTOCOL

Provides the protocol of the server.

Methods

ERROR

Inside a Task, the ERROR procedure immediately fails the Task and passes a custom error message back to the client.

- For synchronous tasks, the HTTP response JSON will contain a jobStatus of "esriJobFailed". Additionally, the jobErrorMessage will be set to the custom *errorMessage*.
- For asynchronous tasks, the job resource JSON will also have the same fields set appropriately.

Syntax

!SERVER.error, *errorMessage*

Arguments

errorMessage

A string that is a customized error message sent to the client.

FILEPATH

The FILEPATH function returns a fully qualified path to the requested file.

Note: FILEPATH does not check to see if the resource exists.

Syntax

Result = !SERVER.filepath('Filename' [, /DATA | /DOCROOT | /LOCAL | /TASK])

Arguments

Filename

A string or an array of strings indicating the file or resource you want to access.

Keywords

The following mutually exclusive keywords are accepted:

- **DATA:** Indicates the filename is under the DATA directory of the server.
- **DOCROOT:** Indicates the filename is under the docroot of the server.
- **LOCAL** (Default): Indicates the filename is under the working directory of the current job. This will change for every request.
- **TASK:** Indicates the filename is under the task directory of the currently running task.

FILETOURL

The FILETOURL function returns a fully qualified URLs to the requested file. The default type of result is a hash or a list of hashes with the key set to 'url' and value set to the fully qualified URL.

Note: FILETOURL does not check to see if the resource exists.

Syntax

```
Result = !SERVER.filetourl('Filename' [, /DATA | , /DOCROOT | , /LOCAL ])
```

Arguments

Filename

A string or an array of strings indicating the file or resource you want to access.

Keywords

The following mutually exclusive keywords are accepted:

- **DATA:** Indicates the filename is under the DATA directory of the server.
- **DOCROOT:** Indicates the filename is under the docroot of the server.
- **LOCAL** (Default): Indicates the filename is under the working directory of the current job. This will change for every request.

NOTIFY

Use the NOTIFY procedure to update the server on the progress of a particular Task. This procedure is used inside asynchronous Tasks to communicate their progress with the server and the client.

The NOTIFY procedure writes the progress value and message to the `gp_job.json` file for a particular Task. The client can then query this file in order to retrieve the Task's status. Additionally, the Jobs Tab of the Admin Console queries the `gp_job.json` file and displays the latest progress value and message for the Task.

Note: If you are developing ESE Tasks in the IDL Workbench, the !SERVER.notify method prints output to the IDL Console if you do not have a "real" server.

Syntax

```
!SERVER.notify, Value, Text
```

Arguments

Value

Set this argument to a user-defined value representing the percent or progress completed. This value can be a number between 1 and 100 but is not tied to a particular data type.

Text

Set this argument to any necessary user-defined information, such as a short progress or phase description, which accompanies the progress update.

URLTOFILE

The URLTOFILE function returns a fully qualified string or string array to the specified resource.

Syntax

Result = !SERVER.urltofile(*URL*)

Arguments

URL

The URL of the resource you wish to access. Both HTTP and file schemes are supported. If using the file scheme, the server must be able to access the location as well.

URL can be the following types:

- **HASH, ORDEREDHASH, and DICTIONARY:** Key must be defined as 'URL' where the value is defined as the URL to convert.
- **LIST:** Must contain only hashes of the type above.

Instantiate a Server

Before you upload a task to ESE, you will likely want to debug it to make sure it works as expected. When debugging your task, you may not have a server running or it may be more convenient to run locally. In cases such as this, the !SERVER system variable and its associated directory structure may not exist. To get around this issue, use the VERIFYSERVER procedure in your task code.

Include the following line at the top of your PRO code to create a !SERVER object and its associated local directory structure:

```
VERIFYSERVER
```

This procedure does the following:

- If not running on a server, or !SERVER doesn't exist, this procedure instantiates the !SERVER system variable and creates its associated directory structure on your local machine (DATADIR, LOCALDIR, DOCROOT, etc.) in its IDL \TMP directory (see the IDL online help documentation on the FILEPATH function for more information). This allows you to debug your task locally before uploading it to the ESE server.

- If running on a server, the `VERIFYSERVER` procedure will quietly do nothing since ESE automatically instantiates a `!SERVER` object for each task to use.

You do not need to remove the `VERIFYSERVER` line the before uploading the task - `VERIFYSERVER` checks to see if `!SERVER` has been instantiated. If it has been instantiated (as it would be on an ESE server), the procedure will do nothing.

ENVI Routines

ESE tasks written in IDL can be used to call ENVI routines. Refer to the "Programming" section of ENVI Help for a list of ENVI routines that you can incorporate into your IDL code.

When a worker process initiates, ESE automatically makes the following call to start ENVI and load all of the components needed to process ENVI commands:

```
e = ENVI (/HEADLESS)
```

A recommended approach is to control ENVI starting and closing through the ESE configuration file.

The /HEADLESS property causes ENVI to start without the user interface.

If ENVI was initiated automatically or by an explicit call within a task, reissuing the above command does not create a second, simultaneous instance of ENVI. Instead, it returns a reference to the existing ENVI session. Only one ENVI session is allowed at any given time.

You can close a worker's instance of ENVI by issuing the following command:

```
e.Close
```

Data Types

See the following topics:

- IDL and ESE Data Types
- ENVI Data Types
- Esri® Data Types

Version History

1.0	Introduced
5.2	Added the following ENVI data types: ENVICoordSys ENVIGridDefinition ENVIPseudoRasterSpatialRef ENVIRasterSeries ENVIRPCRasterSpatialRef ENVISTandardRaster
5.2, Service Pack 1	Added ENVITiePointSet

IDL and ESE Data Types

ESE supports the following data types, which means that if a parameter is published as one of these types, then ESE provides validation and appropriate type casting before sending the parameter to IDL. If a specified data type does not match one of these, then the value is passed directly to IDL and normal dynamic typing applies.

Type	IDL Definition	IDL Conversion
json		Passed to IDL as a string, or passed from IDL (in the form of JSON-serialized data) as a return value back to ESE. The task creator should use JSON_PARSE to validate and retrieve IDL variables.
color	A 3- or 4-byte array	
boolean	A single value or array of values, <code>true</code> or <code>false</code> (not surrounded with quotes)	Converted to IDL BYTE, <code>true</code> =1B, <code>false</code> =0B
byte	An 8-bit unsigned integer (or array of integers) ranging in value from 0 to 255. Pixels in images are commonly represented as byte data.	"B" added on end to cast to IDL BYTE.
int	A 16-bit signed integer (or array of integers) ranging from -32,768 to +32,768	
uint	A 16-bit unsigned integer (or array of integers) ranging from 0 to 65535	
long	A 32-bit signed integer (or array of integers) ranging in value from -2 147 483 648 to +2 147 483 647	"LL" added on end to cast to IDL LONG64, anything after decimal point is dropped, 1e3 -> 1000LL.
ulong	A 32-bit unsigned integer (or array of integers) ranging in value from 0 to 4 294 967 296	
long64	A 64-bit signed integer ranging in value from -9 223 372 036 854 775 808 to +9 223 372 036 854 775 807	
ulong64	A 64-bit unsigned integer (or	

Type	IDL Definition	IDL Conversion
	array of integers) ranging in value from 0 to 18 446 744 073 709 551 615	
float	A 32-bit, single-precision, floating-point number (or array of numbers) of $\pm 10^{38}$, with approximately six or seven significant digits	"D" added on end to cast to IDL DOUBLE, 1e3 -> 1d3. Exponential notation with "e" is changed to "d".
double	A 64-bit, double-precision, floating-point number (or array of numbers) in the range of $\pm 10^{308}$ with approximately 15 or 16 significant digits.	
string	A sequence of characters that is interpreted as text. You can also specify an array of strings.	Surrounded with single quotes, inner quotes escaped The string should be URL encoded. A plus sign (+) in the query portion of a URL string represents the <space> character. To add a literal plus sign into a URL, encode it as: %2B Pound signs (#) in a URL string separate anchors. To encode a literal pound sign into a URL, encode it as: %23

ENVI Data Types

The following ENVI types are treated as JSON objects and validated for proper JSON. They are passed to ESE tasks as strings such that you can call `JSON_PARSE` and get a hash of well-defined key/value pairs to use in the ENVI application programming interface (API):

- `ENVIURI`
- `ENVIRaster`
- `ENVIGCPSet`
- `ENVROI`
- `ENVISpectralLibrary`
- `ENVITiePointSet`
- `ENVIVector`
- Spatial reference data types:
 - `ENVICoordSys`
 - `ENVIGridDefinition`
 - `ENVIPseudoRasterSpatialRef`
 - `ENVIRPCRasterSpatialRef`
 - `ENVISTandardRasterSpatialRef`

For more information on ENVI application programming interface (API) objects, please refer to the ENVI Help.

ENVIURI

This data type is a string that references a local or remote file source. See Options for Selecting Input Data for examples.

ENVIRaster

This data type describes ENVI raster (image) objects.

Key	Description
factory	Required. Specify one of the following strings: ■ <code>URLRaster</code>: Use when opening a new raster. ■ <code>SubsetRaster</code>: Use when defining a spatial or spectral subset.

Key	Description
	■ ResampleRaster: Use when creating a resampled raster. See JSON Syntax below for examples.
bands	Use only when <code>factory</code> is set to <code>SubsetRaster</code>. Use this key to define a subset of bands to process. Specify a scalar or array of zero-based integers corresponding to band numbers. The order of bands is important. Example for bands 1-3 (in that order): <pre>"bands" : [0, 1, 2],</pre>
column_mapping	Use only when <code>factory</code> is set to <code>ResampleRaster</code>. Use in conjunction with <code>row_mapping</code>. If you set this key, do not set <code>pixel_scale</code> or <code>dimensions</code>. Specify an array of numbers that indicate the column index locations in the resampled raster, for each column of pixels in the source raster. The array must have the same number of elements as the source raster's number of columns. If you set this key/value pair but not <code>column_mapping</code>, then no vertical scaling will occur.
dataset_index or dataset_name	Use only when <code>factory</code> is set to <code>URLRaster</code>. This key pertains to <i>container</i> data formats that consist of multiple rasters per dataset. Examples include Landsat ETM+, Landsat-8, NITF, and HDF5. Use the <code>dataset_index</code> key to define a zero-based integer index corresponding to the raster to open. <pre>"dataset_index": 2,</pre> Or, use the <code>dataset_name</code> key and provide a custom string that will be passed in as the DATASET_NAME keyword to <code>ENVI::OpenRaster</code>. Example from an AVHRR NetCDF-4 file: <pre>"dataset_name": "aerosol_dynamics_indicator",</pre>
dimensions	Use only when <code>factory</code> is set to <code>ResampleRaster</code>. If you set this key, do not set <code>pixel_scale</code> or <code>column row_mapping</code>. Specify a two-element array of positive integers that indicate the pixel dimensions of the resampled raster. This provides exact geographic coverage as the source raster.
format	Use only when <code>factory</code> is set to <code>URLRaster</code>. Specify a string that indicates the format of the raster defined by <code>url</code>. Example: <code>"format": "eos_modis"</code>, The following strings indicate formats that correspond to the <code>EXTERNAL_</code>

Key	Description
	TYPE keyword values in ENVI::OpenRaster. If you are opening one of these raster formats, you must specify the appropriate string: acres, ads40, alos, arcview, atsr, avhrr, avhrr_sharp, bmp, cadrg, cib, cosmo-skymed, dmc, dmsp_noaa, ecw, envisat, eol_hdf, eos_aster, eos_modis, er_mapper, erdas_lan, eros_lla, eros_l2b, ers, formosat-2, irs_fast, irs_super_structured, jers, landsat_ceos, landsat_fast, landsat_hdf, landsat_mrlc, landsat_nlaps, master, pci, pds, pict, png, radarsat, seawifs, sicd, spot_cap_sisa, spot_geospot, spot_vegetation, srf, tims, topsar, usgs_doq, usgs_drg, usgs_native_dem, usgs_sdts_dem, xwd When opening a custom data format, you must enter a string with a custom name. This string corresponds to the CUSTOM_TYPE keyword in ENVI::OpenRaster. Example: "format" : "MyFormat"
input_raster	Specify any ENVIRaster JSON object definition.
method	Use only when <code>factory</code> is set to <code>ResampleRaster</code>. Specify the resampling method to use. The choices are as follows: ■ <code>nearest_neighbor</code> (default): Uses the nearest pixel without any interpolation. ■ <code>bilinear</code>: Performs a linear interpolation using four pixels to resample. ■ <code>cubic_convolution</code>: Uses 16 pixels to approximate the sinc function using cubic polynomials to resample the image.
pixel_scale	Use only when <code>factory</code> is set to <code>ResampleRaster</code>. If you set this key, do not set <code>dimensions</code> or <code>column row_mapping</code>. Specify a positive number or two-element array of positive numbers that define how each pixel will be resized. If you specify a single value, then the resizing is the same in the x- and y-directions.
row_mapping	Use only when <code>factory</code> is set to <code>ResampleRaster</code>. Use in conjunction with <code>column_mapping</code>. If you set this key, do not set <code>pixel_scale</code> or <code>dimensions</code>. Specify an array of numbers that indicate the row index locations in the resampled raster, for each row of pixels in the source raster. The array must

Key	Description
	have the same number of elements as the source raster's number of rows. If you set this key/value pair but not <code>column_mapping</code> , then no vertical scaling will occur.
<code>spatialref</code>	Use only when <code>factory</code> is set to <code>SubsetRaster</code> . Specify the name of an <code>ENVIPseudoRasterSpatialRef</code> , <code>ENVIRPCRasterSpatialRef</code> , or <code>ENVISTandardRasterSpatialRef</code> JSON object definition.
<code>sub_rect</code>	Required when <code>factory</code> is set to <code>SubsetRaster</code> . Specify a four-element array expressing the spatial range of the data. The array is of the form: <code>[minX, minY, maxX, maxY]</code> ■ <i>minX</i>: First pixel of the columns dimension. If the <code>spatialref</code> key is set, <i>minX</i> refers to the minimum value in the map x-dimension. ■ <i>minY</i>: First pixel of the rows dimension. If the <code>spatialref</code> key is set, <i>minY</i> refers to the minimum value in the map y-dimension. ■ <i>maxX</i>: Last pixel of the columns dimension. If the <code>spatialref</code> key is set, <i>maxX</i> refers to the maximum value in the map x-dimension. ■ <i>maxY</i>: Last pixel of the rows dimension. If the <code>spatialref</code> key is set, <i>maxY</i> refers to the maximum value in the map y-dimension.
<code>url</code>	Use only when <code>factory</code> is set to <code>URLRaster</code> . Specify a uniform resource locator (URL) identifying a raster file or resource to use. Example: <pre>"url" : "http://server:port/ese/data/qb_boulder_msi.dat"</pre> Example of a URL for an array of <code>ENVIRasters</code>: <pre>input_rasters = [{"factory" : "URLRaster", "url" : "http://server:port/mydata/dataset1"}, {"factory" : "URLRaster", "url" : "http://server:port/mydata/dataset2"}]</pre>

JSON Syntax

Items in italics indicate values that you fill in.

```
{
  "factory" : "URLRaster",
  "url" : "url",
  "format" : "format",
  "dataset_index" : index
}
```

Or:

```
{
  "factory" : "SubsetRaster",
```

```

    "input_raster" : ENVIRaster,
    "sub_rect" : [ minX, minY, maxX, maxY ],
    "bands" : [ band0, band1, ..., bandN ]
}

```

Or:

```

{
    "factory" : "ResampleRaster",
    "input_raster" : ENVIRaster,
    "method" : "nearest neighbor" | "bilinear" | "cubic
convolution",
    "pixel_scale" : [ scaleX, scaleY ],
}

```

The following example subsets a raster by pixel area and picks the first band:

```

{
    "factory" : "SubsetRaster",
    "sub_rect" : [ 0,0,100,100 ],
    "bands" : 0,
    "input_raster" :
    {
        "factory" : "URLRaster",
        "url" : "http://server:port/ese/data/qb_boulder_msi.dat",
    }
}

```

ENVIGCPSet

This data type defines a set of ground control points (GCPs) to use for tasks such as RPC Orthorectification. GCPs associate pixels with known geographic coordinates.

Key	Description
factory	Required. Specify a string value of GCPSet.
url	Required. Use this key to specify a uniform resource locator (URL) identifying a GCP file for use in ENVI processing. Example: "url" : "http://server:port/mydata/control.pts"
apply_offset	Use this key to subtract a designated pixel offset from all of the GCPs defined by url. Specify a two-element array with the x (column) and y (row) offsets. You typically only use this key when applying GCPs from a full input image to a spatially subsetted image with tasks such as RPC orthorectification.

JSON Syntax

Values in italics are those that you fill in.

```
{
    "factory", "GCPSet",
    "url" : "url",
    "apply_offset" : [ xOffset, yOffset ],
}
```

ENVIROI

This data type describes ENVIROI (region of interest) objects.

Key	Description
factory	Required. Specify a string value of <code>URLROI</code> .
url	Required. Use this key to specify a uniform resource locator (URL) identifying an ROI file for use in ENVI processing. Example: <code>"url" : "http://server:port/ease/data/qb_boulder_roi.xml"</code>

JSON Syntax

Values in italics are those that you fill in.

```
{
    "factory", "URLROI",
    "url" : "url"
}
```

ENVISpectralLibrary

Key	Description
factory	Required. Specify a string value of <code>SpectralLibrary</code> .
url	Required. Use this key to specify a uniform resource locator (URL) identifying a spectral library file for use in ENVI processing. Example: <code>"url" : "http://server:port/mydata/veg1_dry.sli"</code>

JSON Syntax

Values in italics are those that you fill in.

```
{
    "factory", "SpectralLibrary",
    "url" : "url",
}
```

```
}
```

ENVITiePointSet

This data type defines a set of tie points used for image-to-image registration.

Key	Description
factory	Required. Specify a string value of <code>TiePointSet</code> .
url	Required. Use this key to specify a uniform resource locator (URL) identifying a tie point file for use in ENVI processing. Example: <code>"url" : "http://server:port/mydata/tiepoints.pts"</code>

JSON Syntax

Values in italics are those that you fill in.

```
{
  "factory", "TiePointSet",
  "url" : "url",
}
```

ENVIVector

This data type describes ENVIVector objects.

Key	Description
factory	Required. Specify a string value of <code>URLVector</code> .
url	Required. Use this key to specify a uniform resource locator (URL) identifying a vector file for use in ENVI processing. Example: <code>"url" : "http://server:port/ese/data/rivers.shp"</code>

JSON Syntax

Values in italics are those that you fill in.

```
{
  "factory", "URLVector",
  "url" : "url"
}
```

Spatial Reference Data Types

The following data types pertain to coordinate systems and spatial references in ENVI.

ENVICoordSys

This data type describes ENVICoordSys objects, which contain coordinate system information for raster and vector files.

Specify one of the following keys, not both:

Key	Description
factory	Required. Specify a string value of <code>CoordSys</code> .
coord_ sys_ code	Specify a valid geographic or projected coordinate system code as an integer. Example: "coord_sys_code" : 20354
coord_ sys_ string	Specify a valid geographic or projected coordinate system string. Example: "coord_sys_string" : "PROJCS["AGD_1984_AMG_Zone_54",GEOGCS["GCS_Australian_1984",DATUM["D_Australian_1984",SPHEROID["Australian",6378160.0,298.25]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION["Transverse_Mercator"],PARAMETER["False_Easting",500000.0],PARAMETER["False_Northing",1000000.0],PARAMETER["Central_Meridian",141.0],PARAMETER["Scale_Factor",0.9996],PARAMETER["Latitude_Of_Origin",0.0],UNIT["Meter",1.0]]"

Refer to the `\IDLxx\resource\pedata\predefined` directory of the ENVI installation path for valid coordinate system codes and strings:

- `EnvipeProjcsStrings.txt`: projected coordinate system codes and strings
- `EnvipeGeogcsStrings.txt`: geographic coordinate system codes and strings

JSON Syntax

Values in *italics* are those that you fill in.

```
{
  "factory", "CoordSys",
  "coord_sys_code" : coord_sys_code,
  "coord_sys_string" : "coord_sys_string"
}
```

```
}
```

ENVIGridDefinition

This data type describes ENVIGridDefinition objects, which provide the information needed to georeference rasters to a common coordinate system. ENVIGridDefinition is only a definition, not a spatial reference or raster. You have three options to define a grid:

- Coordinate system + extents + pixel size
- Coordinate system + extents + number of rows + number of columns
- Coordinate system + pixel size + number of rows + number of columns

Key	Description
factory	Required. Specify a string value of <code>GridDefinition</code> .
coord_sys	Required. Specify an <code>ENVICoordSys</code> object that indicates the coordinate system of the GCPs.
extents	Specify the geographic extent of the grid as follows: <code>[xmin, ymax, xmax, ymin]</code> where x and y are map coordinates (x=eastings, y=northings) or geographic coordinates (x=longitude, y=latitude), depending on the coordinate system. The coordinates must be in the same units (degrees, meters, feet, etc.) as the associated coordinate system. If you set this key, you must also specify (1) <code>pixel_size</code> or (2) <code>nrows</code> and <code>ncolumns</code> . Example: <code>"extents" : [257017.6, 7831362.4, 1153892.7, 7270425.0]</code>
pixel_size	Specify a two-element double-precision array with the [x,y] pixel size in the same units as the associated coordinate system. Example: <code>"pixel_size" : [30.0, 30.0]</code>
nrows	Specify the number of rows in the grid. Example: <code>"nrows" : 1024</code>
ncolumns	Specify the number of columns in the grid. Example:

Key	Description
	<code>"ncolumns" : 1024</code>
<code>tie_point_pixel</code>	Specify a two-element double-precision array with the pixel coordinates of the tie point. If you set this key, you must also specify <code>tie_point_map</code> , <code>nrows</code> , <code>ncolumns</code> , and <code>pixel_size</code> . If you do not set this key, the default pixel coordinates are [0,0].
<code>tie_point_map</code>	Specify a two-element double-precision array with the map coordinates of the <code>tie_point_pixel</code> location, as follows: <code>[mapX, mapY]</code> The coordinates must be in the same units (degrees, meters, feet, etc.) as the associated coordinate system. If you set this key, you must also specify <code>nrows</code> , <code>ncolumns</code> , and <code>pixel_size</code> . Example: <code>"tie_point_map" : [257017.6, 7831362.4]</code>

JSON Syntax

Values in italics are those that you fill in.

```
{
  "factory", "GridDefinition",
  "coord_sys" : ENVICoordSys,
  "extents" : [ xmin, ymax, xmax, ymin ],
  "pixel_size" : [ pixelSizeX, pixelSizeY ]
}
```

Or:

```
{
  "factory", "GridDefinition",
  "coord_sys" : ENVICoordSys,
  "extents" : [ xmin, ymax, xmax, ymin ],
  "nrows" : nrows,
  "ncolumns" : ncolumns
}
```

Or:

```
{
  "factory" : "GridDefinition",
  "coord_sys" : ENVICoordSys,
  "pixel_size" : [ pixelSizeX, pixelSizeY ]
  "nrows" : nrows,
  "ncolumns" : ncolumns
}
```

ENVIPseudoRasterSpatialRef

This data type describes ENVIPseudoRasterSpatialRef objects. Pseudo map information refers to a non-standard projection that involves an affine map transformation. When an image is not georeferenced to a standard map projection and you only know the geographic coordinates and map projection of four points in the image, you can define these tie points and use them to warp the image and to calculate approximate geographic coordinates for each pixel. The pixel size varies in the rectified image.

This type of projection contains a high degree of variability and is not geographically accurate; the (x,y) locations in the rectified image are only best guesses.

Set each key to a four-element array specifying a geographic corner, using Geographic Lat/Lon WGS-84 coordinates, in the form [*X Pixel*, *Y Pixel*, *Longitude*, *Latitude*].

Field	Description
factory	Required. Specify a string value of PseudoRasterSpatialRef.
pseudo_geo_point_1	
pseudo_geo_point_2	
pseudo_geo_point_3	
pseudo_geo_point_4	

JSON Syntax

Values in italics are those that you fill in.

```
{
  "factory" : "PseudoRasterSpatialRef",
  "pseudo_geo_point_1" : [ tiePoint1MapX, tiePoint1MapY,
tiePoint1PixelX, tiePoint1PixelY ],
  "pseudo_geo_point_2" : [ tiePoint2MapX, tiePoint2MapY,
tiePoint2PixelX, tiePoint2PixelY ],
  "pseudo_geo_point_3" : [ tiePoint3MapX, tiePoint3MapY,
tiePoint3PixelX, tiePoint3PixelY ],
  "pseudo_geo_point_4" : [ tiePoint4MapX, tiePoint4MapY,
tiePoint4PixelX, tiePoint4PixelY ]
}
```

ENVIRPCRasterSpatialRef

This data type describes ENVIRPCRasterSpatialRef objects. Rational polynomial coefficients (RPCs) are one type of sensor model that use mathematical coefficients to define the physical relationship between image coordinates and ground coordinates. RPCs are not the same as a map projection; rather, they relate pixel locations in an image to the corresponding latitude, longitude, and elevation, using a third-order rational polynomial.

Most modern high-resolution sensors such as WorldView-3 include pre-computed RPCs with the imagery. If your file has RPC information, you can automatically derive RPC-based geolocation information for individual pixels in an image. This method is not as geographically accurate as performing a full orthorectification, but it consumes less memory and disk space.

Field	Description
factory	Required. Specify a string value of <code>RPCRasterSpatialRef</code> .
rpc_ offsets	Specify a five-element, double-precision array that specifies the <i>Line_Offset</i> , <i>Sample_Offset</i> , <i>Latitude_Offset</i> , <i>Longitude_Offset</i> , and <i>Height_Offset</i> values. Example: "rpc_offsets" : [2536.0000, 3003.0000, 32.833100, -117.23340, 33.000000]
rpc_ scales	Specify a five-element, double-precision array that specifies the <i>Line_Scale</i> , <i>Sample_Scale</i> , <i>Latitude_Scale</i> , <i>Longitude_Scale</i> , and <i>Height_Scale</i> values. Example: "rpc_scales" : [2537.0000, 3004.0000, 0.027900000, 0.062500000, 220.00000]
rpc_ line_ num_ coeff	Specify a 20-element, double-precision array of RPC line numerator coefficients.
rpc_ line_ den_ coeff	Specify a 20-element, double-precision array of RPC line denominator coefficients.
rpc_ samp_ num_ coeff	Specify a 20-element, double-precision array of RPC sample numerator coefficients. Example:

Field	Description
rpc_ samp_ den_ coeff	Specify a 20-element, double-precision array of RPC sample denominator coefficients. Example:

JSON Syntax

Values in italics are those that you fill in.

```
{
    "factory" : "RPCRasterSpatialRef",
    "rpc_offsets" : [ Line_Offset, Sample_Offset, Latitude_Offset,
Longitude_Offset, Height_Offset ],
    "rpc_scales" : [ Line_Scale, Sample_Scale, Latitude_Scale,
Longitude_Scale, Height_Scale ],
    "rpc_line_den_coeff" : [ RPC_line_denominator_coefficients ],
    "rpc_samp_num_coeff" : [ RPC_sample_denominator_coefficients ]
}
```

ENVISpatialRef

This data type describes ENVISpatialRef objects, which is the most common form of spatial reference.

You must specify either `coord_sys_code` or `coord_sys_string`, but not both.

Field	Description
factory	Required. Specify a string value of <code>StandardRasterSpatialRef</code> .
coord_ sys_ code	Specify a valid geographic or projected coordinate system code. Example: "coord_sys_code" : 20354
coord_ sys_ string	Specify a valid geographic or projected coordinate system string. Example: "coord_sys_string" : "PROJCS["AGD_1984_AMG_Zone_54",GEOGCS["GCS_Australian_1984",DATUM["D_Australian_1984",SPHEROID["Australian",6378160.0,298.25]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION["Transverse_Mercator"],PARAMETER["False_Easting",500000.0],PARAMETER["False_Northing",1000000.0],PARAMETER["Central_Meridian",141.0],PARAMETER["Scale_Factor",0.9996],PARAMETER["Latitude_Of_Origin",0.0],UNIT

Field	Description
	<code>["Meter",1.0]]"</code>
pixel_size	Specify a two-element double-precision array with the [x,y] pixel size in the same units as the associated coordinate system. Example: <code>"pixel_size" : [30.0, 30.0]</code>
rotation	Specify a double-precision floating-point value indicating the rotation of the image, degrees clockwise from North. Example: <code>"rotation" : 2.0</code>
tie_point_pixel	Required. Specify a two-element double-precision array with the pixel coordinates of the tie point. If you do not set this key, the default pixel coordinates are [0,0].
tie_point_map	Required. Specify a two-element double-precision array with the map coordinates of the <code>tie_point_pixel</code> location, as follows: <code>[mapX, mapY]</code> The coordinates must be in the same units (degrees, meters, feet, etc.) as the associated coordinate system. Example: <code>"tie_point_map" : [257017.6, 7831362.4]</code>

JSON Syntax

Values in italics are those that you fill in.

```
{
  "factory" : "StandardRasterSpatialRef",
  "coord_sys_code" : coord_sys_code,
  "pixel_size" : [ xPixel, yPixel ],
  "rotation" : "rotation",
  "tie_point_pixel" : [ xPixel, yPixel ],
  "tie_point_map" : [ mapX, mapY ]
}
```

Esri® Data Types

ESE supports the following Esri data types:

Type	IDL Conversion
GPBoolean	Converted to IDL BYTE, true=1B, false=0B
GPMultiValue:GPBoolean	(byte[])
GPDate	"LL" added on end to cast to IDL LONG64.
GPMultiValue:GPDate	(long[])
GPDouble	"D" added on end to cast to IDL DOUBLE. Exponential notation with "e" is changed to a "d".
GPMultiValue:GPDouble	(double[])
GPLong	"LL" added on end to cast to IDL LONG64.
GPMultiValue:GPLong	(long[])
GPString	Surrounded with single quotes, inner quotes escaped.
GPMultiValue:GPString	(string[])

The following Esri types are treated as generic JSON objects, validated for proper JSON and passed to IDL as a string. If desired, the task can pass the string to IDL's JSON_PARSE() routine to turn it into an IDL struct or hash.

- GPLinearUnit
- GPMultiValue:GPLinearUnit
- GPFeatureRecordSetLayer
- GPMultiValue:GPFeatureRecordSetLayer
- GPRecordSet
- GPMultiValue:GPRecordSet
- GPDataFile
- GPMultiValue:GPDataFile
- GPRasterData
- GPMultiValue:GPRasterData
- GPRasterDataLayer
- GPMultiValue:GPRasterDataLayer

For more information, see the ArcGIS® Resources web site.

Writing Client Applications

Client applications include web pages, IDL programs, JavaScript programs, or any other type of program that can make HTTP REST calls. ESE does not provide client functionality, leaving it free to work with any type of client application. ESE does, however, include example clients (located in the `/docroot/ease` directory).

The typical workflow for writing client applications is described below.

1. Publish and test the task, then use the Task Request Builder to submit a job.
2. Note the request string from the Task Request Builder.
3. Write the client application code. See IDL Example Client below for an example client application written in IDL.
 - Construct an HTTP request string, using the request string in the Task Request Builder as a pattern.
 - Invoke the IDLnetURL object's "get" method.

HTTP strings typically follow this pattern, where the italicized items are user-specified values:

`http://hostname:port/ease/catalog/service/task/operation?query string`

The following is an example of the format for the `ese_addition` task below:

```
http://enghost:8181/ease/services/SyncService/ese_
addition/execute?a=2&b=2
```

In this example:

- `enghost` is the hostname of the master node.
- `8181` is the port number.
- `ease` is the main ESE endpoint.
- `services` is the catalog name.
- `SyncService` is the service. Services contain one or more related tasks of the same synchronicity.
- `ese_addition` is the task name.
- `execute` is the operation.
- `?a=2&b=2` is the query string. The input parameters in the example are `a` and `b`.

Variable	Value	Description
<catalog>	services	Hard-coded
<service>	SyncService AsyncService	Hard-coded

Variable	Value	Description
<task>	<i>IDL routine name</i>	The name of the task to run.
<operation>	execute submitJob	
<query string>	<key value> <key value> & <query string>	Leave empty when the task does not have input parameters.
<key value>	<key> = <value>	
<key>	<i>variable name</i>	The name of a variable matching one in the task's *.task.
<value>	<boolean> <null> <string> <number> <json>	
<boolean>	true false	
<null>	null	
<string>	<i>string</i>	Single quotes, double quotes, or neither.
<number>	<i>number</i>	64-bit integers and floating-points.
<json>	<i>JSON</i>	See JSON specification.

For more examples on selecting input data and executing tasks, see Options for Selecting Input Data.

Example IDL Client

The following example uses the `ese_addition` task to add two numbers. It uses the `IDLnetURL` object to perform the HTTP requests. Find the example with comments in the examples directory of your ESE installation.

```

;+
; NAME:
; ese_addition_example
;
; PURPOSE:
; Provides an example of calling a synchronous ENVI Services Engine
task
;   from IDL PRO code.
;
;   Before running the example, do the following:
;
; (1) Upload the "ese_addition" task to the server. Do this by
using the
;   admin console to upload the "ese_addition.zip" file.
;   (2) Update this program with the proper server and port.

```

```
;
; MODIFICATION HISTORY:
; 2012-11-19: DRE: initial write.
;-

PRO ese_addition_example
COMPILE_OPT IDL2

    ; specify host and port
    ;
    ; The host and port identify a running instance of the ENVI
    Services
    ; Engine. Change these parameters to match the settings in
    ; <ESE_WORK_DIR>/bin/server.cfg.

    host = 'penny64'
    port = '8181'

    ; add two numbers ( a + b = ? )
    ;
    ; The IDLnetURL object enables IDL to be a client to an HTTP
    server.
    ; It will hit the same "ese_addition" service that the web page
    example
    ; client uses. The admin console's task request builder shows us
    that
    ; we want a request string that looks like:
    ;
    ; http://penny64:8181/ese/services/SyncService/ese_
    addition/execute?a=1&b=2
    ;
    ; Note that we are leaving off an optional third parameter. That
    is
    ; left as an exercise for the reader.

    oNetURL = IDLnetURL()

    a = 1
    b = 2

    url = 'http://' + host + ':' + port + $
    '/ese/services/SyncService/ese_addition/execute?' + $
    'a=' + STRTRIM( a, 2 ) + '&' + $
    'b=' + STRTRIM( b, 2 )

    result = oNetURL.get( url = url, /STRING_ARRAY )
```

```
OBJ_DESTROY, oNetURL

; get the sum
;
; The "get" returned the result in JSON format. Since /STRING_
ARRAY
; was specified, the string array is first converted to a single
string.
;
; Note: to get an accurate idea of what the JSON will look like,
run the
; admin console's Task Request Builder on the task you're
interested in.
; Then inspect the resultant JSON.
;
; The JSON is then parsed into nested IDL hash and list objects.
Behold
; the power of hash and list indexing.

json = JSON_PARSE( STRJOIN( result ) )

; Index into a mix of hashes and lists.
value = JSON[ 'results', 0, 'value' ] ; index into a mix of
hashes and lists

; Print the computation.
PRINT, STRTRIM( a, 2 ) + ' + ' + STRTRIM( b, 2 ) + ' = ' +
STRTRIM( value, 2 )
END
```

Options for Selecting Input Data

This topic provides examples of various schemes used to input data to run tasks in the ENVI service endpoint. ESE supports the `http:`, `jpeg:`, and `file:` schemas. Task developers can use the `!SERVER.URLToFile` system variable in PRO code to support these schemes.

Replace italicized words below with actual values from the end user. Also update `localhost` to the server name that is running ESE.

Local file:

These options only work if the client and ESE are on the same machine.

Examples of parameters that require a data type of `ENVIURI`:

```
http://localhost:8181/ease/services/AsyncService/ConvertToPNG/submitJob?INPUT_FILE="file:///C:/ese/qb_boulder_pan"
```

```
http://localhost:8181/ease/services/AsyncService/ConvertToPNG/submitJob?INPUT_FILE="file:/share/data/qb_boulder_pan"
```

Example of a parameter that requires a data type of `ENVIRaster`:

```
http://localhost:8181/ease/services/ENVI/ColorSliceClassification/submitJob?INPUT_RASTER={"url":"file:///C:/ese/qb_boulder_pan","factory":"URLRaster"}&REVERSE_COLOR_TABLE=false
```

Network file:

This option only works if the server is on a Windows platform; the operating system of the client does not matter. The end user running the service must be able to access the UNC path.

```
http://server:8181/ease/services/ENVI/ColorSliceClassification/submitJob?INPUT_RASTER={"url":"file://share/data/qb_boulder_pan","factory":"URLRaster"}&REVERSE_COLOR_TABLE=false
```

The following option only works if the server is on a Linux platform. The operating system of the client does not matter.

```
http://server:8181/ease/services/ENVI/ColorSliceClassification/submitJob?INPUT_RASTER={"url":"file:/share/data/qb_boulder_pan","factory":"URLRaster"}&REVERSE_COLOR_TABLE=false
```

JPIP stream:

```
http://
server
:8181/ease/services/ENVI/ISODATAClassification/submitJob?CHANGE_
THRESHOLD_PERCENT=2.0000000&INPUT_RASTER=
{"url":"jpip://server/jp2/utm.jp2","factory":"URLRaster"}
&ITERATIONS=10&NUMBER_OF_CLASSES=5
```

Esri® Image Services stream:

```
http://
server
:8181/ease/services/ENVI/ISODATAClassification/submitJob?CHANGE_
THRESHOLD_PERCENT=2.0000000&INPUT_RASTER=
{"url":"http://exampleserver:1234/arcgis/rest/services/folder_
name/dataset_name/ImageServer/catalogId","factory":"URLRaster"}
&ITERATIONS=10&NUMBER_OF_CLASSES=5
```

OGC stream:

```
http://
server
:8181/ease/services/ENVI/ISODATAClassification/submitJob?CHANGE_
THRESHOLD_PERCENT=2.0000000&INPUT_RASTER={"url":"http://ogc-
server/cgi-bin/mapserv.exe?map=/ogc_
data/map&Coverage=imageName&service=wcs","factory":"URLRaster"}
&ITERATIONS=10&NUMBER_OF_CLASSES=5
```

ESE:

```
http://
server
:8181/ease/services/ENVI/ColorSliceClassification/submitJob?INPUT_
RASTER={"url":"http://server:8181/ease/data/qb_boulder_
pan.dat","factory":"URLRaster"}&REVERSE_COLOR_TABLE=true
```

Knowledge Base

This topic provides answers to frequently asked questions, as well as troubleshooting steps. See the following sections:

- Installation
- System Administration
- Tasks and Clients

Installation

Please refer to the Exelis VIS Help Articles web site for up-to-date information on installation and licensing. Here are some frequently asked questions with regard to installation:

- **Is it possible to run ESE in a Windows Virtual Machine on a Linux machine?** Yes, this works in ESE.
- **Can ENVI LiDAR be installed and used with ESE?** Yes, ENVI LiDAR can be installed with ESE (on Windows). Follow the instructions in the installer. This combined Windows installation enables processing services that use both the ENVI API and the ENVI LiDAR API.

System Administration

Frequently Asked Questions

- **Are there any tips for starting or restarting ESE?** Any time you restart ESE, you must start the master node first before attempting to start any of the worker nodes. The master node maintains the licensing for the workers, so the workers must be able to communicate with the master in order for the license transfer to take place.
- **Are there any guidelines for the number of concurrent workers?** Our experiments have shown that a good practice is to have the number of ESE workers equal to half of the number of cores. For example, if you have an 8 core machine, then choose to run 4 workers. This is the default in the configuration file. This is independent of hyperthreading. Even one IDL or ENVI task can fully utilize an entire multi-core machine. The likelihood of this happening depends on the functions used in the algorithms (some IDL functions are "threaded" while some are not.) Set the number of workers (per machine) accordingly and set the threading level of IDL and ENVI accordingly.

- If you have tasks that are highly disk intensive, then it may be okay to run workers at a one-to-one ratio with logical cores. If you have tasks that are highly CPU intensive, then we suggest scaling back the number of workers. It's also important to keep in mind that some IDL or ENVI tasks can utilize all cores on a system by virtue of the IDL thread pool and ENVI NCOM library (which uses the Intel TBB library).
- **How do I rename services back to their original names?**
 1. Open the Admin Console and navigate to the Tasks tab.
 2. On left portion of the screen, in the folder navigation interface, click on the service whose name you want to change (for example, "AsyncService").
 3. Right-click on its name and choose "Edit Properties" to open the Properties dialog.
 4. In the dialog, delete the existing Service Name ("AsyncService" in our example) and type in the original name (for example, "GPAsync").
 5. Click **OK**.
 6. Your updated service name should now appear in place of its ESE name in the Admin Console.

Troubleshooting

I cannot access the Admin Console

1. Check for firewall issues that may prevent you from accessing the server. Adjust your settings to allow the appropriate port.

Port 80 is the default http port on most installations, and is normally opened by the firewall. On Linux, a process must be owned by root in order to use port 80. You can use one of the following commands to have the kernel redirect tcp traffic on port 80 to another port, where ESE is not running as root.

If iptables is available:

```
sudo /sbin/iptables -t nat -I PREROUTING -p tcp --dport 80 -j  
REDIRECT --to-port 8181
```

If ipchains is available:

```
sudo /sbin/ipchains -I input --proto TCP --dport 80 -j  
REDIRECT 8181
```

2. Allow remote connections from the master machine.
3. Check for permission problems. For example, a possible scenario is that the user is on a laptop that is not connected to the correct network. Perhaps the

laptop is being used remotely, but has not connected to the authenticating VPN. The user can log in, but is not fully authenticated. This would leave the ESE processes without the proper credentials to access ports, such as 8181. The solution is to make sure ESE is running as a properly privileged user and that the user is fully authenticated, perhaps by logging into the VPN.

4. Check to see what user installed ESE. If you install ESE while logged in as `root` (not just `sudo` the installer, but are actually logged in as `root` when you install), you will not be able to run ESE while logged in as a different user. When you install ESE as `root`, the `root` user owns all of the working files that ESE uses and they cannot be modified by a lesser privileged user. One fix for this issue is to change ownership of all the files in the `<ESE_WORK_DIR>` by using the `chown` command. The correct way to handle the issue is to install ESE as the user you want to run as.
5. Check the license file on the server to see if it has expired. You can open the file in a text editor to check its expiration date. See Licensing ESE for more information.

The server will not start

If a server fails to start, look for information in the log files contained in `<ESE_WORK_DIR>/logs`. Possible problems include:

- `server.cfg` errors
- license failures

Run ESE as an application to more visibly see the errors.

The service does not start in a Windows clustered environment

Check to see if the `<ESE_WORK_DIR>` is accessed using UNC syntax.

By design in Windows, a service cannot directly access a mapped network drive. But as part of the ESE installation process in a Windows Clustered Environment, the `<ESE_WORK_DIR>` must be installed on a network or shared drive. If you attempt to install the `<ESE_WORK_DIR>` on a mapped network drive, the ESE service will not start.

The recommended configuration for Windows clusters is to use UNC syntax when setting the location of the `<ESE_WORK_DIR>` during installation, i.e.,

`\\<ComputerName>\<SharedFolder>\<ESE_WORK_DIR>`.

If a drive letter is necessary in your situation, creating a directory symbolic link may also work. For example, from Windows, run `cmd.exe` as admin and enter:

```
mklink /d <link> <target>
```

Where `<link>` specifies the new symbolic link name and `<target>` specifies the path (relative or absolute) the new link references.

For more information on Windows services, UNC, and symbolic links, see:

- <http://support.microsoft.com/kb/180362>
- http://en.wikipedia.org/wiki/Uniform_Naming_Convention#Uniform_Naming_Convention
- http://en.wikipedia.org/wiki/NTFS_symbolic_link

The service does not start on Windows after a password change

If the password for the user associated with the ESE service changes, it needs to be changed in the service manually in order for the ESE service to start. To do this, select **START > Control Panel > System and Security > Administrative Tools > Component Services > Services Local**. Then right click on the service name and select **Properties**. Next click the Log On tab and change the username/password for the specified account.

An error is issued when starting the service on RedHat Enterprise5

If you receive the following error message:

```
/usr/local/exelis/idl8x/bin/bin.linux.x86_64/EnviServicesEngine:
error while loading shared libraries:
/usr/local/exelis/idl8x/bin/bin.linux.x86_64/libcurl.so.3: cannot
restore segment prot after reloc: Permission denied
```

You can start the service by using the following command:

```
/usr/sbin/setenforce 0
```

A more permanent option is to set the SE option to "disabled" in the file:

```
/etc/selinux/config.
```

This error occurs because RedHat and CentOS use SE Linux by default, which blocks the service from starting. Note that this is not an issue in Debian-based distributions.

The daemon does not start when my Linux machine boots

First, check that ESE was set to run as a daemon. There are a couple of ways to do this. On RedHat-based systems, the following command indicates that the daemon was properly set up:

```
% chkconfig --list ese
```

Note that `chkconfig` can often be found in `sbin` and `update-rc.` in `/usr/sbin`. The output should be something like:

```
ese 0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

On RedHat and Debian systems, the following command lists out the system's links to the ESE daemon script for each run level:

Red Hat	Debian
<code>find /etc/rc.d -name '*ese' -print</code>	<code>find /etc/rc*.d -name '*ese' -print</code>

The output should be something like:

Red Hat	Debian
<code>/etc/rc.d/rc4.d/S50ese</code>	<code>/etc/rc0.d/K20ese</code>
<code>/etc/rc.d/rc0.d/K50ese</code>	<code>/etc/rc1.d/K20ese</code>
<code>/etc/rc.d/rc3.d/S50ese</code>	<code>/etc/rc2.d/S80ese</code>
<code>/etc/rc.d/rc6.d/K50ese</code>	<code>/etc/rc3.d/S80ese</code>
<code>/etc/rc.d/rc1.d/K50ese</code>	<code>/etc/rc4.d/S80ese</code>
<code>/etc/rc.d/init.d/ese</code>	<code>/etc/rc5.d/S80ese</code>
<code>/etc/rc.d/rc2.d/S50ese</code>	<code>/etc/rc6.d/K20ese</code>
<code>/etc/rc.d/rc5.d/S50ese</code>	

The entries with an "S" indicate that the daemon will be started at that run level and entries with a "K" indicate that the daemon will be killed at that particular run level.

Now, check your current run level by running the command:

```
% who -r
```

The most common run levels are 3 and 5. Regardless, the current run level should be "on" or in the list of `rc*.d` directories with an "S". If the ESE daemon script does not appear to be set to start at the desired run level, then run the following commands:

Red Hat	Debian
<code>sudo chkconfig --del ese</code>	<code>sudo update-rc.d -f ese remove</code>
<code>sudo chkconfig --add ese</code>	<code>sudo update-rc.d -f ese start 80 2 3 4 5 . stop 20 0 1 6 .</code>
<code>sudo chkconfig --level 2345 ese on</code>	

In the last command, replace the run levels with whatever run levels are desired. The effect can be verified by running the commands listed at the top of this troubleshooting section.

If the first set of efforts (described above) fail, then you can try a more complete removal and re-installation. Run the following commands to first uninstall the ESE daemon script and then reinstall it.

Red Hat	Debian
<code>sudo chkconfig --del ese</code>	<code>sudo update-rc.d -f ese remove</code>
<code>sudo rm /etc/init.d/ese</code>	<code>sudo rm /etc/init.d/ese</code>
<code>cd <ESE_WORK_DIR>/bin</code>	<code>cd <ESE_WORK_DIR>/bin</code>
<code>sudo ese-install-initd -u <user></code>	<code>sudo ese-install-initd -u <user></code>

The last command runs an Exelis VIS-supplied script that installs and configures the ese daemon script into `/etc/init.d` and calls `chkconfig` or `update-rc.d` to add that script as a daemon, including all the links in the `/etc/rc.d` subdirectories. Once those commands are run, re-check the daemon setup by running the commands in the first part of this troubleshooting section. If all looks good then try rebooting the machine and testing ESE.

If the ESE daemon still does not start when the operating system boots, please contact Exelis VIS Technical Support.

Tasks and Clients

My task does not upload

Verify that the IDL code is correct and the `*.task` (or `config.json` in ESE 1.0.1 and 5.1) file is valid. (Validate it with this JSON validator: <http://jsonlint.org>.) Check the error log in the Admin Console (Logs tab) for more information.

My tasks do not run

Verify these settings in the configuration file:

- Does the `master` parameter specify a valid machine name that you can ping?
- Do all of the nodes in the network that are running ESE processes know about this machine name?
- Does the `listening_port` parameter specify a valid port that is opened on your firewall?

Check your directories to make sure they have write permissions by the user ID running your service:

- `/docroot`
- `/data`
- `/logs`
- `/tasks`

If a task does not show in the Available Tasks List, check the following:

- Is the task file named correctly? It must be `*.task`.
- Is the task file formatted correctly? Make sure it has the correct commas and parentheses in must have a comma on each line ending (except for opening brackets).
- Make sure that the task `.pro` and `*.task` files are bundled and zipped in a folder with the name of the task (the same as the `.pro` code file).

If a task shows in the Available Task List but still will not run correctly:

- Check the log for the job that failed for detailed information and IDL output.
- For positional `InvocationType`, the order of parameters listed in the `*.task` file must match the order of the arguments in IDL routine. Optional parameters must come *after* required ones.

Contact Us

Corporate Office

Exelis Visual Information Solutions, Inc., a subsidiary of Harris Corporation
4990 Pearl East Circle
Boulder, CO 80301 USA
Web: <http://www.exelisvis.com>
E-Mail: info@exelisvis.com
Phone: 303-786-9900
Fax: 303-786-9909

Technical Support

E-Mail: support@exelisvis.com
Phone: 303-413-3920

Training

E-mail: training@exelisvis.com

Sales and Custom Development

E-Mail: sales@exelisvis.com